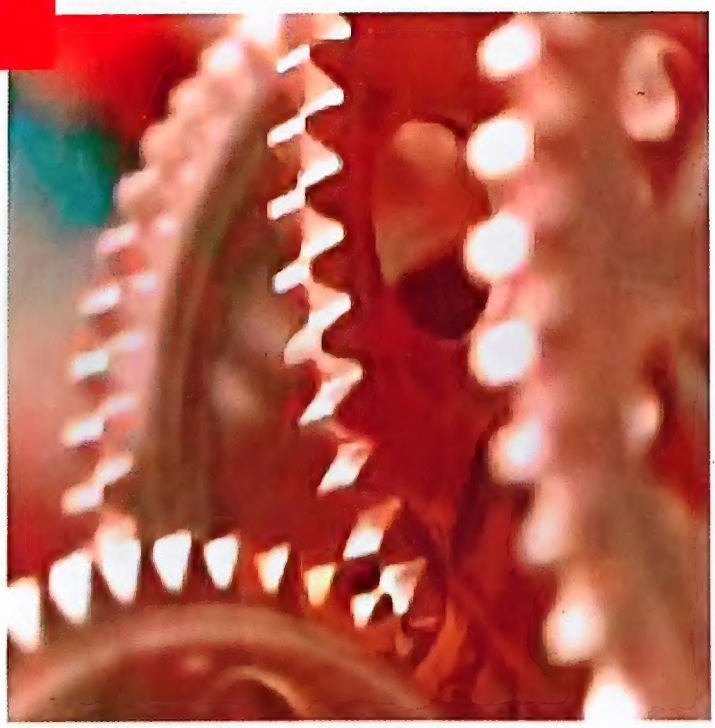
profesional
fundamente

Marin Fotache

Proiectarea bazelor de date

Normalizare și postnormalizare

Implementări SQL și Oracle



POLIROM

MARIN FOTACHE

Proiectarea bazelor de date

Normalizare şi postnormalizare. Implementări SQL şi Oracle

Marin Fotache (n. 1966) este absolvent al Universităţii „Al.I. Cuza" din Iaşi, Facultatea de Economie şi Administrarea Afacerilor, specializarea Finanţe şi Contabilitate. Este profesor la catedra de Informatică economică (*www.feaa.uaic.ro*). Dintre lucrările publicate: *SQL. Dialecte DB2, Oracle şi Visual FoxPro* (Polirom, Iaşi, 2001), *Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale* (în colaborare, Polirom, Iaşi, 2002); *Oracle 9i2. Ghidul dezvoltării aplicaţiilor profesionale* (în colaborare, Polirom, Iaşi, 2003); este şi autorul a numeroase articole de specialitate publicate în revista *NetReport*.

© 2005 by Editura POLIROM

www.polirom.ro

Editura POLIROM
Iaşi, B-dul Carol I nr. 4, P.O. BOX 266, 700506
Bucureşti, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33, O.P. 37; P.O. BOX 1-728, 030174

Descrierea CIP a Bibliotecii Naţionale a României:

FOTACHE, MARIN

Proiectarea bazelor de date: normalizare şi postnormalizare: implementări SQL şi Oracle / Marin Fotache. - Iaşi: Polirom, 2005

ISBN: 973-681-898-5

004.65
004.43 SQL
004.43 ORACLE

Printed in ROMANIA

Marin Fotache

Proiectarea bazelor de date

Normalizare și postnormalizare

Implementări SQL și Oracle

POLIROM
2005

Tablă de materii

Introducere 9

CAPITOLUL 1 - **Obiectul proiectării bazelor de date** 13
1.1. Locul bazelor de date în sistemul informaţional 13
1.2. Principalele arhitecturi ale aplicaţiilor în care sunt implicate bazele de date 16
1.3. O metodologie clasică de dezvoltare a aplicaţiilor 19
1.4. Privire de ansamblu asupra proiectării bazelor de date 25
1.5. Proiectarea bazelor de date în prezentul volum 29

CAPITOLUL 2 - **La început a fost normalizarea** 31
2.1. Concepte ale modelului relaţional de organizare a bazelor de date 32
2.1.1. Relaţii/tabele, domenii, atribute, valori nule 33
2.1.2. Restricţii ale bazei de date 35
2.2. Nevoia de normalizare 39
2.2.1. Redundanţe 40
2.2.2. Anomalii la inserare 40
2.2.3. Anomalii la modificare 40
2.2.4. Anomalii la ştergere 41
2.3. Normalizarea - câteva tuşe 41
2.4. Etapele normalizării 42
2.5. Descompunere fără pierdere de informaţii 44
2.6. Relaţia universală 46

CAPITOLUL 3 - **Prima formă normalizată** 51
3.1. Definiţii ale primei forme normalizate 51
3.2. Atribute simple şi atribute compuse 53
3.3. Despre grupuri repetitive şi urmările lor 54
3.3.1. Grupuri repetitive pe orizontală 55
3.3.2. Eliminarea grupurilor repetitive prin adăugarea de tupluri 57
3.3.3. Eliminarea grupurilor repetitive prin construirea de noi relaţii 61
3.4. Prima formă normală şi problema cheii primare a relaţiei universale 67
3.5. Dezbateri pe marginea şi în centrul primei forme normale 76

CAPITOLUL 4 - **Dependenţe funcţionale. A doua formă normalizată** 83
4.1. Dependenţe funcţionale 83
4.1.1. Definiţie 83

4.1.2. Dependenţe funcţionale cu sursa compusă 86
4.1.3. Proprietăţi ale dependenţelor funcţionale, închidere şi acoperire minimală 91
4.1.4. Dependenţe funcţionale simetrice şi redundante 93
4.2. Reprezentări ale dependenţelor funcţionale 94
4.2.1. Diagrame ale DF 94
4.2.2. Graful dependenţelor funcţionale 96
4.2.3. Matricea dependenţelor funcţionale 98
4.3. Dependenţe funcţionale totale 100
4.4. Trecerea relaţiilor în a doua formă normală (2NF) 101

CAPITOLUL 5 - **A treia formă normalizată şi forma normală Boyce-Codd** 111
5.1. Dependenţe funcţionale directe/tranzitive 111
5.2. Aducerea relaţiilor în a treia formă normală 114
5.3. Aducerea relaţiilor în 3NF prin grafuri şi matrice ale DF 120
5.4. Forma normală Boyce-Codd 123
5.5. Câteva probleme ale normalizării 130
5.5.1. DF aparent tranzitive, dar care conţin informaţii neredundante 131
5.5.2. Dependenţe tranzitive necanonice 132
5.5.3. Relaţii semantice imposibil de preluat în dependenţe funcţionale 138
5.5.4. Simetrie şi temporalitate în dependenţele funcţionale 143
5.5.5. Grafuri şi BCNF 146

CAPITOLUL 6 - **Dependenţe multivaloare şi de joncţiune. Formele normalizate 4 şi 5** 149
6.1. Dependenţe multivaloare 149
6.2. A patra formă normală 156
6.3. Dependenţe de joncţiune 164
6.4. A cincea formă normală 170
6.5. Alte forme normale 171

CAPITOLUL 7 - **Surogate, reguli, incluziuni. Soluţii SQL şi Oracle. Un caz practic** 173
7.1. Chei surogat 173
7.1.1. Nevoia de surogate 173
7.1.2. Puncte de vedere privind cheile surogat 177
7.1.3. Declararea/obţinerea cheilor surogat 180
7.2. Restricţii simple şi complexe. Atribute redundante ajutătoare 181
7.2.1. Reguli la nivel de atribut şi înregistrare 182
7.2.2. Reguli, atribute redundante şi declanşatoare 183
7.2.3. Restricţii şi relaţii noi 191
7.3. Dependenţe de incluziune 194
7.4. Caz practic - centru de închiriere a casetelor video 205

CAPITOLUL 8 - **Generalizări, ierarhii, grafuri, echivalenţe** 231
8.1. Generalizare şi specializare 231

8.2. Ierarhii generalizate 241
8.3. Grafuri 253
8.3.1. Distanţe între localităţi. Trasee 254
8.3.2. Generarea traseelor 258
8.3.3. Rezervări şi gestiunea călătoriilor 261
8.4. Incluziuni & echivalenţe 266

CAPITOLUL 9 - **Denormalizare şi postnormalizare** 269
9.1. Denormalizarea - rău-rău, dar chiar necesar? 269
9.2. Categorii de operaţii legate de denormalizare 272
9.3. Copii „exacte" ale unui atribut în mai multe tabele 277
9.4. Atribute derivate/sintetice 279
9.5. Relaţii redundante/sintetice 297
9.6. Atribute introduse pe baza grupurilor repetitive 307
9.7. Ruperea relaţiilor pentru creşterea vitezei de acces 311

CAPITOLUL 10 - **Temporalitate şi acces în baze de date** 313
10.1. Validitatea temporală a unei scheme 313
10.2. Consemnarea evoluţiei unor parametri. „Refacerea" unei valori anterioare 320
10.3. Protejarea „trecutului". Închideri contabile şi nu numai 335
10.4. Arhivări, fuziuni, compatibilităţi 339

Bibliografie 345

Introducere

După a doua carte publicată la Polirom (*Visual FoxPro...*, 2002), un cititor se declara amuzat de frământările mele şi ale colegilor în a găsi un titlu adecvat fiecărei cărţi, titlu care să atragă posibilul cumpărător şi să semnaleze cât mai limpede ceea ce se ascunde între copertele cărţii. Amuzamentul său, de altminteri şi al altor cititori care ne-au scris ulterior, ţinea de faptul că *frământările* ce, pe alocuri, degenerau în *frăsuieli*, au eşuat în nişte titluri cu totul convenţionale, atât pentru *SQL*, cât şi pentru *VFP*. Mă tem că aceasta este şi soarta prezentului volum, din care vor lipsi însă, cel puţin din introducere, *frăsuielile*.

Teritoriul proiectării bazelor de date, revendicat de analişti/proiectanţi, dar „călcat", cu sau fără nonşalanţă, şi de dezvoltatorii de aplicaţii, administratorii de baze de date sau de alte categorii de informaticieni, este unul în care talentul „urechist" (în sensul de „cântăreţ/interpret-care-nu-are-nevoie-de-partitură") a jucat multe feste în istoria recentă a proiectelor informaţionale de la noi şi de aiurea.

Motivul principal al masivei doze de improvizaţie şi eşecuri în proiectarea bazelor de date ţine de însăşi natura demersului. Fără a exagera grosolan, putem afirma că, în construirea schemei unei baze de date, rigoarea este pozitivă doar până la un punct. Intuiţia, inteligenţa (nici aceasta în doze prea mari, deoarece strică!), dar mai ales experienţa sunt calităţi de neînlocuit ale unui „gânditor" într-ale bazelor de date. Prin comparaţie cu administrarea şi cu scrierea de programe şi proceduri ce alcătuiesc „logica bazei de date", proiectarea este zona cea mai puţin rigidă, cea mai inexactă.

Al doilea motiv este, în mare măsură, o consecinţă a primului: lipsă cronică de cărţi şi monografii despre cum se gândeşte schema unei baze de date. Practic, chiar pe plan mondial, cărţile cu adevărat valoroase dedicate proiectării bazelor de date pot fi numărate pe degete (fără a le folosi pe toate!). Şi aici aş furniza o explicaţie simplistă. Academiştii, adică universitarii precum autorul de faţă, au răbdare să adune sistematic, şi mai ales să compileze din alte lucrări (fără a le indica neapărat la bibliografie), chestiuni ce sunt relevante şi/sau la modă, însă, de obicei, evită cu dibăcie să se implice în proiecte practice, aplicative. Senzaţia pe care am încercat-o citind majoritatea cărţilor dedicate bazelor de date este că autorii nu au lucrat la nici un proiect de anvergură în domeniu. Pe de altă parte, cei care s-au desăvârşit în proiectarea bazelor de date (incomparabil mai puţini decât cei ce cred acest lucru) câştigă, de obicei, destul de binişor şi-şi cultivă alte plăceri decât cea a scrisului, cu atât mai mult cu cât cronofagia proiectelor informaţionale afectează iremediabil

condiţia intelectuală, cârpită disperat cu o lectură din Patapievici şi o alta din Cărtărescu[1].

După cum, probabil, cei mai atenţi dintre dvs. şi-au dat seama, am adoptat tactica lui Ion Iliescu aplicată la Revoluţie şi după. Am început prin a descrie situaţia ca pe una disperată, aproape fără ieşire, tocmai pentru a demonstra cât de importanţi suntem noi în salvarea „situaţiunii". Aşa că (urmează o tuse scurtă, în doi timpi, *à la* Cristian Greţcu de la Divertis), *concetăţeni şi concetăţene*, iată ce urmează:

Mai întâi, vom cheltui câteva pagini cu importanţa bazelor de date pentru sistemele informaţionale ale organizaţiilor şi a locului „dânselor" în aplicaţiile software. Metodologia clasică de dezvoltare a aplicaţiilor este completată, în paragrafele 1.4 şi 1.5, de un adevărat *lamentatio* al proiectării bazelor de date, care este destinat a lămuri ceea ce individualizează această carte faţă de celelalte lucrări consacrate subiectului.

În al doilea capitol vom introduce noţiunile pe care se bazează modelul relaţional şi apoi vom face o schiţă, în tuşe grosolane, a normalizării, pornind de la nevoia de normalizare şi terminând cu etapele procesului.

Din capitolul trei discuţia se animă un pic, deoarece intrăm în câteva zone scandaloase ale normalizării: valori atomice, grupuri repetitive, relaţii universale şi prima formă normală (1NF). Lucrurile sunt cu atât mai interesante cu cât chiar în tabăra „marilor preoţi" ai relaţionalului sunt câteva subiecte de gâlceavă, iar o parte dintre definiţii au fost reformulate de-a lungul ultimilor ani.

Discuţia animată din capitolul trei devine de-a dreptul plicticoasă în cel de-al patrulea, deoarece aici vor fi tratate dependenţele funcţionale, împreună cu proprietăţile acestora, precum şi modalitatea de aducere a relaţiilor în a doua formă normalizată (2NF). Din păcate, ideea de a sări peste acest capitol nu-i tocmai bună, deoarece întreg procesul de normalizare se bazează pe dependenţele dintre atribute.

Capitolul cinci acoperă un obiectiv intermediar în construirea schemei unei baze de date, obiectiv care, pentru unii teoreticieni şi destui practicieni, este chiar final: a treia formă normalizată (3NF). Lucrurile nu se opresc aici, deoarece 3NF este completată cu discutarea formei normale Boyce-Codd (BCNF) şi a câtorva dintre situaţiile în care normalizarea îşi arată limitele sau este de-a dreptul neputincioasă.

Capitolul şase încearcă să facă mai atractivă, nu fără o anumită doză de zădărnicie, probabil cea mai deprimantă zonă a normalizării, cea referitoare la dependenţele multivaloare şi de joncţiune şi, implicit, la formele normale 4 şi 5 (4NF şi 5NF). Iar dacă pentru 5NF practica nu oferă prea multă satisfacţie dornicilor de exemple, problemele legate de 4NF, deşi ascunse cu grijă sub preş, sunt mult mai frecvente decât ne-am dori. Pe final sunt pomenite câteva tentative de depăşire a 5FN, cu precădere DKNF, care însă nu au trecut de stadiul bunelor intenţii.

În capitolul şapte intrăm deja în „sinergia faptelor". Discuţiile coboară nepermis de la fineţuri şi abstracţiuni teoretice la probleme mundane, cum ar fi cheile surogat, restricţiile complexe, atributele redundante şi dependenţele de incluziune. De fapt,

1. Părerea mea!

dacă ne place să ne jucăm cu cuvintele, putem spune că în capitolul 3 recepţionam baza de date *nenormalizată*, din capitolul 3 în capitolul 6 am tot *normalizat-o*, iar din acest capitol intrăm în *post-normalizarea* bazei de date care, în capitolul 9, va căpăta pe alocuri forma *denormalizării*. Cazul practic din final este gândit pentru a răzbuna cantitatea de teorie ingurgitată în precedentele capitole. Tot ca element de noutate, din acest capitol apar mostre de cod scrise aproape exclusiv în Oracle PL/SQL. *Listing*-urile şi procedurile sunt gândite ca exemple de implementare a chestiunilor teoretice prezentate.

Capitolul 8 stă sub semnul exoticului, cel puţin pentru o lucrare dedicată normalizării şi post-normalizării. Deşi considerate cvasi-monopol al orientării pe obiect, generalizarea şi specializarea pot fi abordate parţial, dintr-un unghi propriu, şi în modelul relaţional, iar scurta şi incompleta discuţie despre (un soi de) grafuri ne duce cu gândul la modelul reţea, astăzi pe drept uitat de cei mai tineri dintre noi. Finalul capitolului, dedicat reprezentării simultane a incluziunilor şi echivalenţelor într-o schemă, nu ne duce cu gândul la nimic.

Dintr-o carte în care normalizarea este la aşa mare preţ nu putea lipsi un capitol dedicat *denormalizării*, ce poate fi privită şi ca o încălcare parţială (şi uşor jenată) a aproape tot ceea ce am discutat în capitolele anterioare. Folosită deseori abuziv, denormalizarea poate fi asimilată mai degrabă postnormalizării, deci nu e atât o încălcare a normalizării, cât o completare a sa.

Una dintre cele mai dure probe ale unei baze de date este validitatea în timp. Tocmai de aceea capitolul 10 tratează, pe scurt, câteva aspecte ale temporalităţii în proiectarea schemei unei baze de date. Ne interesează deci să găsim acele structuri de date care să furnizeze toate informaţiile importante ale aplicaţiei, indiferent de momentul producerii lor, dar şi mecanismele de protejare a datelor sensibile împotriva alterării lor ulterioare.

Prin ce s-ar diferenţia de concurenţă cartea de faţă? În primul rând, prin tentativa de a umaniza un domeniu considerat imposibil – normalizarea. Arma cel mai des întrebuinţată este exemplul şi contraexemplul. Cu riscul de a părea dezlânat, materialul de faţă este înţesat de cazuri practice care ilustrează aproape fiecare noţiune discutată. Un al doilea atu al lucrării ţine de mostrele de cod care aduc discuţiile legate de proiectare pe teritoriul implementării bazei de date. Prin toate mecanismele şi artificiile descrise, redactate în SQL sau Oracle PL/SQL, se doreşte creşterea credibilităţii cărţii, ştiindu-se că un dezavantaj curent al publicaţiilor dedicate proiectării ţine de apetenţa lor exclusiv teoretică. În al treilea rând, cartea răstoarnă canoanele clasice ale proiectării unei baze de date, întrucât renunţă (de bună voie şi nesilită de nimeni) la modelele consacrate ale analizei şi proiectării sistemelor informaţionale: diagrame entităţi-relaţii, diagrame UML etc. De fapt, nu renunţă la acestea, cât amână discuţia pentru un volum viitor. Pur şi simplu, în acest volum am vrut să epuizăm cât mai mult din ceea ce este relevant pentru practicieni din normalizare şi postnormalizare.

Să publici în 2005 o carte despre proiectarea bazelor de date fără o brumă de E-R, UML, O-R, OO pare desuet, dacă nu chiar sinucigaş (publicistic). Ei bine, citiţi dvs.

cartea de faţă, şi peste doi-trei ani ne întâlnim la o bârfă de UML. N-aş vrea să se înţeleagă că se poate renunţa la analiză şi proiectare, că dezvoltarea de aplicaţii complexe ar trebui să se bazeze exclusiv pe normalizare. Am vrut să arăt doar că normalizarea este mult mai generoasă decât am înţeles noi de la teoreticienii săi, că este un demers bazat pe concubinajul rigorii cu ambiguitatea, că există situaţii pe care normalizarea le rezolvă magistral şi situaţii în care este de-a dreptul penibilă.

Rămâne de văzut şi cât anume din intenţii are acoperire în ceea ce se aşterne în faţa dvs. În ceea ce mă priveşte, ceea ce urmează reprezintă nu numai fructul compilaţiei cărţilor şi articolelor la care am avut acces în ultimii cincisprezece ani, ci şi experienţa, uneori traumatizantă, a câtorva zeci de proiecte şi aplicaţii în care am fost implicat, chiar dacă, nu rareori, rezultatele au fost dezastruoase. Dintre colegii de catedră, îi mulţumesc îndeosebi lui Florin pentru răbdarea de a fi citit şi criticat prima formă a manuscrisului şi, totodată, îi cer scuze pentru necazul de a-i răpi din timpul afectat treburilor casnice (scuze, Florin, n-am putut să mă abţin!).

Marin Fotache
Iaşi, februarie 2005

Notă:
Listing-urile SQL şi PL/SQL din capitolele 7-10 pot fi descărcate de pe pagina Web a Facultăţii de Economie şi Administrarea Afacerilor, Universitatea „Al.I. Cuza", Iaşi: *http://www.feaa.uaic.ro/cercetare/publicatii*.

CAPITOLUL 1

Obiectul proiectării bazelor de date

De la portofoliul de megaaplicaţii circumscrise avalanşei acronimice în care este cu neputinţă să nu facem confuzii – ERP, SCM, CRM, BI, B2B, B2C etc. – până la „anticele" (în terrneni strict informatici) şi pernicioasele aplicaţii de contabilitate generală scrise în FoxPro 2.0, vorbim de sisteme (sau fragmente) informaţionale, de porţiuni ale memoriei organizaţionale. Putem caracteriza bazele de date ca un ingredient esenţial, ce-i drept, mai puţin vizibil, al sistemului informaţional al oricărei firme, organizaţii, instituţii.

Tocmai caracterul oarecum discret, dacă nu cumva misterios a făcut din bazele de date un domeniu interesant şi atractiv pentru profesioniştii IT, dar a generat şi numeroase confuzii, convulsii, bonjurisme, partizanate şi idiosincrasii. Cele câteva tuşe din acest capitol, dedicate bazelor de date şi abordărilor privitoare la proiectarea bazelor de date, nu vor tranşa nicidecum disputele, dar nici nu le vor acutiza.

1.1. Locul bazelor de date în sistemul informaţional

Mai toate lucrările de profil includ datele, alături de echipamente (hardware), programe (software), proceduri, metodologii şi personal, drept componente esenţiale ale sistemelor informaţionale în organizaţii.

Putem privi bazele de date pieziş sau cu simpatie, din diferite unghiuri, în funcţie de interes, pregătire de specialitate şi competenţă în domeniu. Înainte de a îneca subiectul în câteva definiţii curat academice menite a semăna panică printre studenţi, trebuie să spunem că mulţi dintre cei dispuşi sau siliţi să fie preocupaţi de domeniul IT privesc bazele de date ca pe nişte rezervoare imense în care sunt stocate informaţii dintre cele mai diverse, informaţii ce ulterior pot fi extrase, prelucrate, sistematizate după dorinţele şi nevoile diverselor tipuri de utilizatori.

Analogia cu rezervoarele funcţionează până la un punct. În primul rând, rezervoarele trebuie croite corespunzător, pentru a nu pierde din caracteristicile substanţelor stocate, pentru a putea face sintetizările (distilările) ulterioare şi pentru a nu fi în situaţia de a extinde ulterior, din mers, încăperi ale rezervoarelor deja în uz (vorbim de o schemă judicioasă de stocare). În al doilea rând, n-ar strica să existe câteva filtre care să blocheze introducerea de substanţe „cu defecte", întrucât mai toată lumea cunoaşte principiul informatic cu numele GIGO – *Garbage In, Garbage Out*,

adică „Gunoi introduci, gunoi extragi ! " (aici am putea vorbi despre un mecanism de integritate a bazei de date). Pe urmă, introducerea substanţelor în rezervor, ca şi extragerea lor, brută sau prin rafinare, necesită utilaje speciale (acesta este sistemul de gestiune a bazei de date) şi angajaţi recomandabil pricepuţi (aici intră, de-a valma, proiectanţi, administratori de baze de date, programatori, utilizatori mai răsăriţi în ale bazelor de date etc.).

Aducând discuţia pe făgaşul academic, o bază de date (BD) reprezintă un ansamblu structurat de fişiere care grupează datele prelucrate în aplicaţiile informatice ale unei persoane, grup de persoane, întreprinderi, instituţii etc. Formal, BD poate fi definită ca *o colecţie de date aflate în interdependenţă, împreună cu descrierea datelor şi a relaţiilor dintre ele* sau, similar, *o colecţie de date utilizată într-o organizaţie, colecţie care este automatizată, partajată, definită riguros (formalizată) şi controlată la nivel central*[1]. Într-o abordare mai analitică, o BD este un ansamblu de date *structurate*, *coerente*, *neredundante*, *independente de orice program de aplicaţie*, *direct accesibile după mai multe criterii*.

O bază de date evoluează în timp, în funcţie de volumul şi complexitatea proceselor, fenomenelor şi operaţiunilor pe care le reflectă. Ansamblul informaţiilor stocate în bază la un moment dat constituie *conţinutul*, *instanţierea* sau *realizarea* acesteia. Organizarea bazei de date se reflectă în *schema* sau *structura* sa, ce reprezintă un ansamblu de instrumente pentru descrierea datelor, a relaţiilor dintre acestea, a semanticii lor şi a restricţiilor la care sunt supuse. În timp ce volumul prezintă o evoluţie spectaculoasă în timp, schema unei baze rămâne relativ constantă pe tot parcursul utilizării acesteia.

Cu bazele de date lucrează o generoasă tipologie de profesionişti IT şi utilizatori, lucru vizibil şi din clasica schemă din figura 1.1, ce ilustrează cele trei niveluri la care se poate discuta despre schema şi conţinutul unei baze de date : *fizic* sau intern, *conceptual* sau global şi *extern*.

La nivel *fizic* (intern), structura datelor este descrisă foarte detaliat, fiind accesibilă numai specialiştilor (ingineri de sistem, programatori în limbaje de asamblare sau alte limbaje apropiate de „maşină"). Esenţiale pentru acest nivel sunt fişierele stocate în memoria externă a calculatorului, fişiere ce conţin datele propriu-zise, alcătuite din articole sau înregistrări cu format comun.

Nivelul *conceptual* (sau *global*) este nivelul imediat superior celui fizic, datele fiind privite prin prisma semanticii lor ; interesează conţinutul lor efectiv, ca şi relaţiile care le leagă de alte date. Obiectivul acestui nivel îl constituie modelarea realităţii considerate, asigurându-se independenţa bazei faţă de orice restricţie tehnologică sau echipament anume. Toţi utilizatorii îşi exprimă nevoile de date la nivel conceptual, prezentându-le administratorului bazei de date, acesta fiind cel care are o viziune globală necesară satisfacerii tuturor cerinţelor informaţionale.

Nivelul *extern* este ultimul nivel de abstractizare la care poate fi descrisă o bază de date. Structurile de la nivelul conceptual sunt relativ simple, însă volumul lor

1. Pentru amatorii de definiţii, vezi [Lungu ş.a. 95], p. 14 ; [Băscă 97], p. 11 ; [Popescu 97], pp. 12-14 ; [Velicanu ş.a. 99], pp. 7-8 ; [Fotache ş.a. 02-2], p. 227 ; [Fotache 01], p. 12 ; [Fotache 97], p. 29.

poate fi deconcertant. Iar dacă la nivel conceptual baza de date este abordată în ansamblul ei, în practică, un utilizator sau un grup de utilizatori lucrează numai cu o porţiune specifică a bazei, în funcţie de departamentul în care îşi desfăşoară activitatea şi de atribuţiile sale. Simplificarea interacţiunii utilizatori-bază, precum şi creşterea securităţii bazei sunt deziderate ale unui nivel superior de abstractizare, care este nivelul extern. Astfel, structura BD se prezintă sub diferite machete, referite uneori şi *ca sub-scheme*, scheme externe sau imagini, în funcţie de nevoile fiecărui utilizator sau grup de utilizatori.

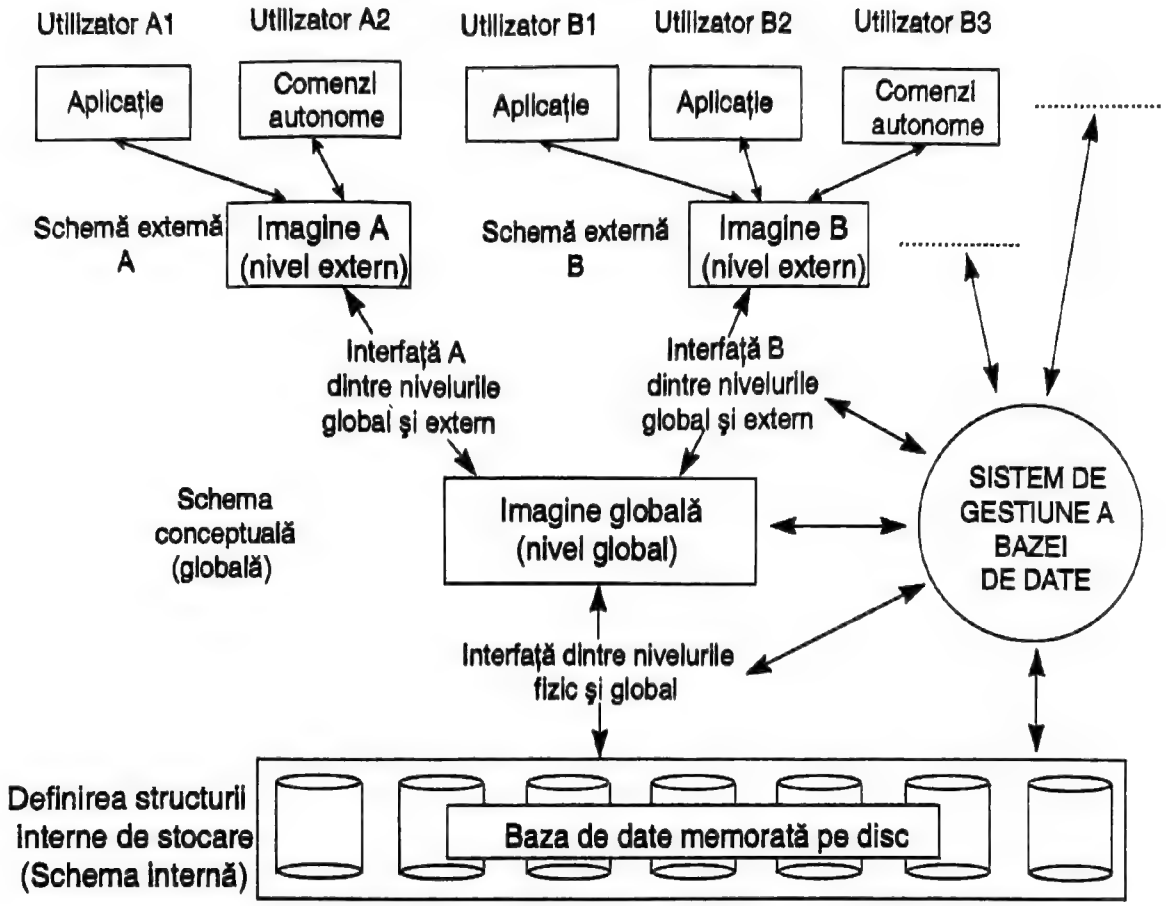


Figura 1.1. Schematizarea unui sistem de lucru cu o bază de date[2]

Accesul utilizatorilor la informaţiile din bază este posibil numai prin intermediul componentei software reprezentate de sistemul de gestiune a bazei de date (SGBD). În general, interfaţa utilizator-SGBD se poate realiza în două moduri: printr-un mecanism de apel inserat în programele scrise într-un limbaj „tradiţional", acesta fiind cazul SGBD-urilor cu limbaj gazdă, sau prin comenzi speciale utilizate autonom (în afara aplicaţiilor-program), în cazul SGBD-urilor autonome.

2. Prelucrare a schemei din [Date 86], p. 33.

1.2. Principalele arhitecturi ale aplicaţiilor în care sunt implicate bazele de date

Una dintre proprietăţile esenţiale ale bazelor de date ţine de caracterul lor public, de accesibilitatea lor, e drept, diferenţiată, pentru o largă tipologie de utilizatori. Rareori se întâmplă ca baza de date să fie pe acelaşi calculator de pe care utilizatorul o accesează. Lucrul acesta este valabil doar pentru aplicaţiile simple, *monoutilizator*, cum sunt referite în folclorul informatic. De obicei însă, baza de date se află pe un calculator special sau chiar pe mai multe calculatoare, dacă vorbim de baze de date replicate sau distribuite.

Dacă sistemul pe care se află baza de date este folosit doar la stocarea acesteia, iar SGBD-ul care o exploatează este instalat pe fiecare calculator client, se poate vorbi de o arhitectură *file-server*. Iniţial, reţelele de PC-uri se bazau pe descărcarea pe staţia de lucru a fişierelor de pe server şi efectuarea tuturor prelucrărilor local. Problema de bază a unei asemenea arhitecturi era incapacitatea deservirii simultane a unui număr mare de utilizatori.

Ca răspuns la limitările conceptului *file-server* a apărut arhitectura *client-server*, în care serverul de fişiere a fost înlocuit cu un server de baze de date pe care este instalat un SGBD care „ascultă" cererile clienţilor şi le răspunde în mod direct. Arhitectura client-server se constituie ca o infrastructură versatilă, bazată pe mesaje, modulară, născută cu intenţia îmbunătăţirii utilizabilităţii, flexibilităţii, interoperabilităţii şi scalabilităţii aplicaţiilor software. Clientul este definit ca un solicitant de servicii, iar serverul – ca un furnizor de servicii, ţinând seama de faptul că una şi aceeaşi staţie poate fi şi client, şi server, în funcţie de configuraţia software. Clientul şi serverul comunică prin schimb de mesaje[3].

În general, clienţii sunt calculatoare personale (PC-uri) utilizate pentru activităţi de gestionare a datelor. După Sinha[4], un post *client* se caracterizează prin faptul că:

a) prezintă o interfaţă utilizator, care e, de obicei, grafică (GUI);
b) „formulează" interogări (cereri, consultări) sau comenzi pe care le „înaintează" serverului;
c) transmite interogările/comenzile respective serverului prin intermediul unei tehnologii de comunicaţie;
d) analizează datele din rezultatele interogărilor/comenzilor primite de la server.

Iar un *server* se defineşte prin faptul că:

a) furnizează un serviciu clientului;
b) răspunde la interogările/comenzile clientului;
c) ascunde detaliile sistemului, făcând transparent dialogul dintre client şi server.

3. Vezi [Fotache ş.a. 02-2].
4. Sinha, R., „Client/server computing", *Communications of the ACM*, iulie 1992, pp. 77-98.

Serverele pot fi calculatoare personale sau sisteme de calcul specializate (minicalculatoare, *mainframe*-uri) în vederea asigurării legăturii dintre clienţi şi bazele de date din care se extrag informaţiile dorite. Dezideratul este autonomia informatică a fiecărui angajat, în limita unor atribuţii delegate, angajat care poate astfel consulta şi prelucra datele din orice loc aflat în interiorul sau în exteriorul întreprinderii. Prin accesul la toată gama serviciilor disponibile în reţea se poate reacţiona mai rapid şi mai eficace la evenimentele care apar şi deci exercita mai eficient responsabilităţile încredinţate.

Orice sistem client/server este alcătuit din minimum trei componente principale: *interfaţa cu utilizatorul* (*sistem de operare/mediu grafic*), *aplicaţia* (*prelucrările sau procesele*) şi *sistemul de gestiune a bazelor de date*. În practică, există mai multe modalităţi de repartizare a funcţiilor între client şi server, aşa cum rezultă şi din figura 1.2[5].

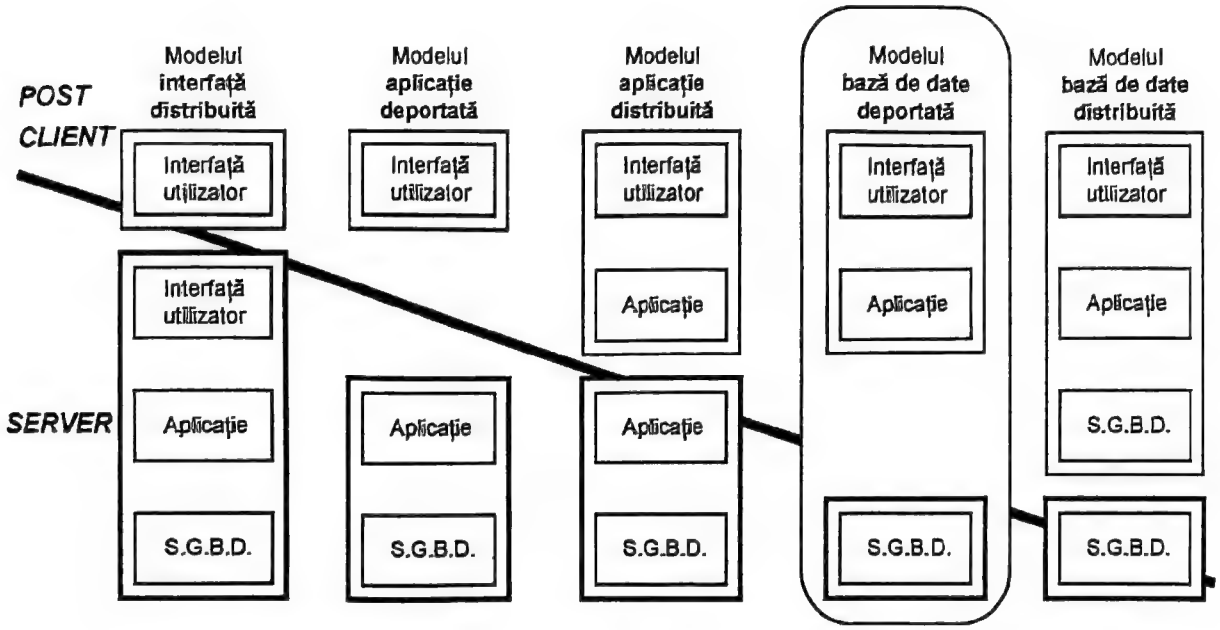


Figura 1.2. Cinci moduri de repartizare a funcţiilor între client şi server (*sursa:* Gartner Group)

- *Modelul interfaţă distribuită.* Codul aferent interfeţei utilizator este împărţit între platformele client şi server, iar aplicaţia şi SGBD-ul sunt ambele rezidente pe server. Este modelul unei arhitecturi clasice cu un calculator central gazdă la care sunt cuplate terminale pasive (sistemele de minicalculatoare româneşti INDEPENDENT, CORAL, mini-sistemele PDP ale firmei DEC etc.).
- *Modelul aplicaţie deportată.* Codul aferent interfeţei este plasat pe platforma client, în timp ce aplicaţia şi SGBD-ul sunt situate pe server (XWindows rulând sub UNIX).
- *Modelul aplicaţie distribuită.* Presupune localizarea interfeţei pe calculatorul client, a SGBD-ului pe calculatorul server, în timp ce o parte a aplicaţiei este rezidentă pe staţie şi cealaltă parte pe server.

5. Preluare din Harmon, P., „Les systèmes client-serveur", *Génie logiciel et systèmes experts*, nr. 34, martie 1994.

- *Modelul bază de date deportată.* Interfaţa şi aplicaţia sunt rezidente pe platforma client, în timp ce SGBD-ul este plasat pe server. Este modelul arhitecturii client-server „clasice", pe logica lui fiind realizate majoritatea sistemelor comercializate sub titulatura „client-server".
- *Modelul bază de date distribuită.* Interfaţa, aplicaţia şi o primă parte a SGBD-ului se găsesc pe sistemul client, în timp ce o a doua parte a SGBD-ului este rezidentă pe server.

În prezent, asistăm la schimbarea paradigmei în dezvoltarea aplicaţiilor, modelul client-server fiind considerat depăşit de mulţi autori, iar marile companii din industria software ne urează „bun venit" în lumea aplicaţiilor Web[6]. În esenţă, arhitecturile pe trei straturi, spre deosebire de arhitecturile client-server pe două straturi, separă *logica afacerii* pe un strat distinct, localizat de regulă pe un server de aplicaţii care comunică strâns cu serverul de baze de date. Paradigma dezvoltării software-ului bazat pe componente ce comunică între ele prin interfeţe standardizate a făcut posibilă creşterea numărului de straturi armonizate în arhitecturi generic numite *n*-tier (vezi figura 1.3).

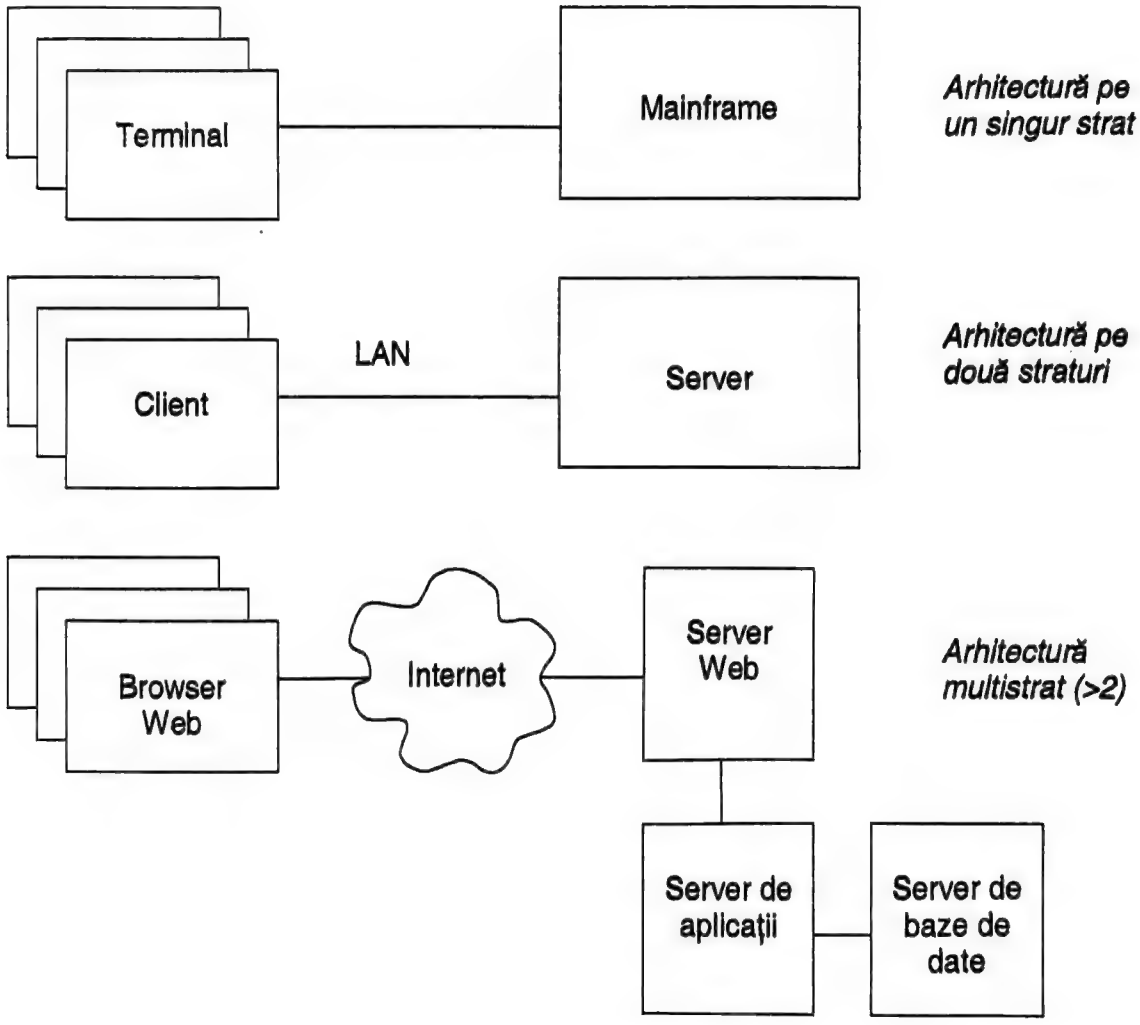


Figura 1.3. Trei tipuri de arhitecturi

6. Vezi [Fotache ş.a. 02-2], pp. 252-275.

Tehnologiile Web permit accesarea aplicaţiilor folosind browsere Web, esenţiale fiind protocoale ca HTML (pentru prezentare) sau HTTP (pentru comunicare între clienţii şi serverele Web).

1.3. O metodologie clasică de dezvoltare a aplicaţiilor

Până acum am punctat importanţa datelor în sistemul informaţional al organizaţiilor, nivelurile la care se poate aborda structura şi conţinutul unei baze de date, precum şi locul „stratului" *date* în aplicaţiile informatice actuale. În acest paragraf vom încerca să încadrăm activitatea (activităţile) de proiectare a bazei de date în demersul mai larg al dezvoltării de aplicaţii.

De la bun început trebuie precizat că subiectul paragrafului de faţă constituie obiect de studiu pentru cel puţin două domenii ale informaticii aplicate în organizaţii, sau, mai bine spus, ale sistemelor informaţionale în organizaţii. Mai întâi, este vorba de ceea ce în literatura anglo-saxonă se numeşte *software engineering*, în franceză *génie logiciel*, iar la noi *dezvoltare de aplicaţii software*. Este un domeniu conturat încă din anii '60 şi aflat de atunci, cel puţin după spusele marilor specialişti, într-o permanentă criză, ce nu dă semne de epuizare.

Al doilea domeniu este mai larg şi ţinteşte mult mai mult decât realizarea de aplicaţii, deşi aceasta reprezintă totuşi un obiectiv central. Este vorba de *analiza şi proiectarea sistemelor informaţionale*[7]. Analiza şi proiectarea se ocupă, printre altele, şi cu investigarea tuturor proceselor, operaţiunilor şi tranzacţiilor dintr-o firmă sau instituţie, cerinţelor utilizatorilor şi perspectivelor organizaţionale, astfel încât toate aceste aspecte să fie luate în calcul în realizarea unui sistem informaţional coerent, aliniat misiunii, obiectivelor şi politicilor firmei. Cu alte cuvinte, analiza şi proiectarea pornesc mai din amonte, de la utilizatori, procese, tranzacţii economice, încercând să formalizeze/modeleze realitatea sub forma unei largi game de diagrame, scheme, specificaţii pe care le vor înainta realizatorilor modulelor de interfaţă, prelucrări şi date.

Una dintre cele mai cunoscute scheme de realizare (dezvoltare) a aplicaţiilor de lucru cu baze de date sau, altfel spus, schemă de principiu a ciclului de viaţă al aplicaţiilor cu BD este cea din ediţiile lucrării lui Thomas Connoly şi Carolyn Begg[8], prezentată în figura 1.4[9].

Amploarea activităţilor din ciclul de viaţă a unei BD depinde de anvergura aplicaţiei. Când sistemul dezvoltat vizează un număr redus de utilizatori şi se referă la un ansamblu de funcţii nu din cale-afară de complex, multe etape sunt sărite sau parcurse sumar.

Planificarea bazei de date

Planificarea BD presupune eşalonarea paşilor ciclului de viaţă pentru atingerea unui maximum de eficacitate. Ca şi în cazul planificării software-ului, planificarea

7. Vezi [Oprea 99].
8. Până la ediţia a treia, apărea şi un al treilea autor, A.D. Strachan – vezi [Connoly ş.a. 96].
9. [Connoly & Begg 02], pp. 271-294.

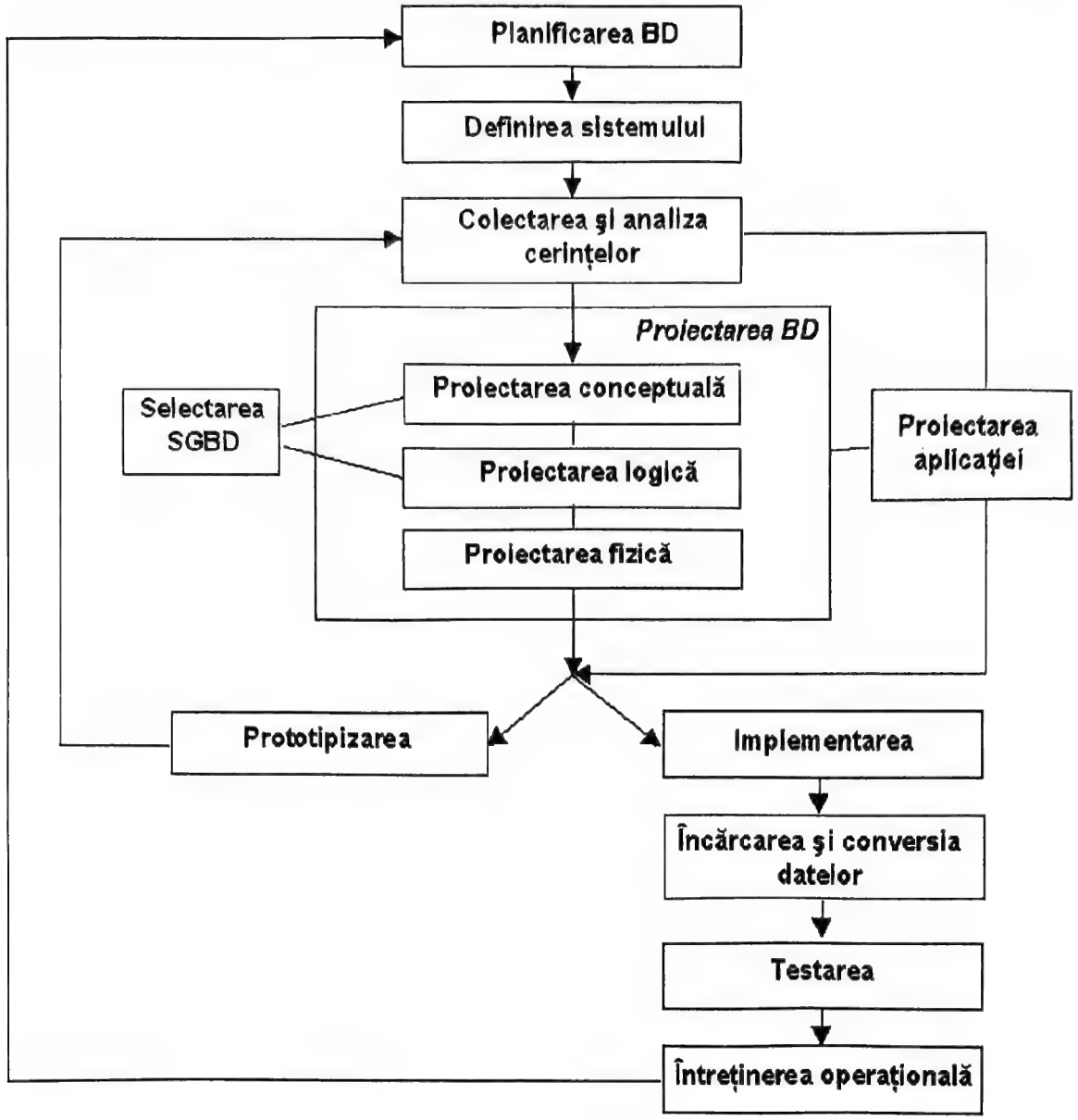


Figura 1.4. Ciclul de viaţă a aplicaţiilor ce utilizează baze de date

BD presupune identificarea şi evaluarea *activităţilor* ce trebuie derulate (întreprinse), *resurselor* necesare derulării activităţilor, *fondului de timp*, *specialiştilor* şi *banilor disponibili*. Planificarea BD trebuie integrată în strategia de ansamblu a firmei, unul dintre obiectivele esenţiale fiind catalizarea activităţilor, politicilor şi strategiei unităţii.

Definirea sistemului

Definirea sistemului presupune specificarea domeniului şi graniţelor aplicaţiei ce lucrează cu baza de date, utilizatorilor, aplicabilităţii sale, precum şi a celorlalte componente ale sistemului informaţional la care se va conecta noul subsistem.

Colectarea şi analiza cerinţelor

Această etapă implică adunarea şi analiza cerinţelor aplicaţiilor din partea utilizatorilor. Cerinţa reprezintă o opţiune, un element ce trebuie inclus, tratat în noul

sistem. Proiectarea BD se bazează pe informaţii despre organizaţii, informaţii ce trebuie preluate şi gestionate de bază. Acestea pot fi culese în diferite moduri:

- intervievarea personalului din întreprindere, cu precădere a celor mai apropiaţi, prin specificul activităţii lor, de profilul aplicaţiei, avându-se în vedere acordarea unei atenţii mărite experţilor din compartimentele funcţionale considerate;
- observarea modului de derulare a operaţiilor în cadrul firmei;
- examinarea documentelor, în principal a celor purtătoare de informaţii (primare, rapoarte);
- folosirea chestionarelor pentru preluarea informaţiilor de la un mare număr de utilizatori;
- utilizarea experienţei acumulate la proiectarea unor sisteme similare.

Informaţia culeasă trebuie să includă: principalele domenii (zone) şi grupuri de utilizatori; documentaţia utilizată sau generată de şi despre aceste domenii/grupuri de utilizatori; detaliile tranzacţiilor reclamate de fiecare domeniu/grup; o listă de cerinţe, pe priorităţi a fiecărui domeniu/grup.

Rezultatul acestei activităţi îl reprezintă *specificaţiile cerinţelor organizaţionale*, prezentate, de obicei, sub forma unui set de documente în care sunt descrise, din diferite unghiuri, operaţiile întreprinderii. De obicei, cerinţele specificate se prezintă într-o formă puţin structurată şi necesită transformarea utilizând tehnici consacrate, cum ar fi cele de tip *SAD – Structured Analisys and Design* (analiză si proiectare structurată), *DFD – Data Flow Diagrams* (diagrame ale fluxurilor de date), diagrame *HIPO – Hierarchical Input Process Output* (ierarhice – intrări-prelucrări-ieşiri) etc.

Proiectarea bazei de date

Aceasta include proiectarea logică şi fizică a BD. În urma acestei etape va fi elaborat modelul BD care va constitui suportul obiectivelor şi operaţiunilor organizaţiei. Principalele obiective ale proiectării BD sunt:

- reprezentarea datelor şi a relaţiilor dintre date, formulare pe diferitele zone ale aplicaţiei şi ale grupurilor de utilizatori;
- furnizarea unui model al datelor care să permită orice tranzacţie autorizată asupra acestora;
- construirea unui proiect care va atinge cerinţele de performanţă ale sistemului, cum ar fi: securitatea, timpul de răspuns etc.

Deseori însă, realizarea unui obiectiv se face în detrimentul altuia. Spre exemplu, un nivel ridicat de securitate atrage după sine scăderea vitezei de răspuns.

După Connoly şi Begg[10], există două abordări majore ale proiectării BD. Abordarea *bottom-up* porneşte de la nivelul elementar, cel al atributelor, care sunt grupate în clase de entităţi şi asociaţii. Normalizarea este un exemplu tipic de demers *bottom-up*, care dă rezultate bune atunci când numărul atributelor nu este prea mare.

10. [Connoly & Begg 02], pp. 279-280.

Când numărul de atribute este ridicat, mai nimerită este abordarea *top-down*, care dezvoltă mai întâi un model sintetic, simplificat al datelor. Cele câteva entităţi sunt descompuse ulterior în mai multe etape, până la identificarea atributelor şi entităţilor elementare. Modelarea folosind diagrame E-R are la bază abordarea *top-down*.

Există şi alte abordări ale proiectării BD ; spre exemplu, *inside-out* (de la interior către exterior) seamănă cu *bottom-up*, dar diferă prin faptul că mai întâi se stabileşte un set de concepte majore şi apoi analiza se lărgeşte prin includerea în model şi a altor concepte care se află în relaţie cu cele majore. *Strategia mixtă* utilizează deopotrivă, pentru diferitele părţi ale modelului, şi *bottom-up*, şi *top-down*, combinând rezultatele.

După cea mai mare parte a autorilor, proiectarea bazelor de date este de două tipuri, *logică* şi *fizică*[11], în timp ce după alţii există trei tipuri : *conceptuală*, *logică*, *fizică*[12].

Selectarea sistemului de gestiune a bazei de date

Considerată un pas opţional, selectarea SGBD presupune alegerea celui mai adecvat SGBD pentru realizarea şi exploatarea aplicaţiei. Nu toate SGBD-urile satisfac cerinţele aplicaţiei, iar, pe de altă parte, cele mai performante sunt şi cele mai scumpe. Cerinţele de funcţionalitate trebuie coroborate cu resursele disponibile, în vederea identificării celui mai adecvat SGBD comercial pentru realizarea şi exploatarea aplicaţiei.

Criteriile de selecţie au fost generos sistematizate de Connoly şi Begg ca în tabelul 1.1[13].

Proiectarea aplicaţiei

Proiectarea aplicaţiei se referă la conceperea secvenţelor de cod (instrucţiuni în limbaje de programare) ce vor lucra cu baza. Din figura 1.4 se observă că proiectarea BD şi proiectarea aplicaţiei sunt activităţi distincte, paralele ale ciclului de viaţă al aplicaţiilor cu BD. În cea mai mare parte a cazurilor, proiectarea aplicaţiei nu poate fi finalizată înainte de proiectarea bazei. Pe de altă parte, informaţiile proiectării vor fi transmise proiectării BD.

În această etapă trebuie verificat dacă toate funcţiile specificate în cerinţele utilizatorului se regăsesc în proiectul aplicaţiei. Proiectarea aplicaţiei include şi activitatea de *proiectare a tranzacţiilor*. De asemenea, tot acum se elaborează şi modelul interfeţei cu utilizatorul a aplicaţiei.

Prototipizarea

Deşi opţională, prototipizarea se poate dovedi deosebit de utilă prin construirea unui model de lucru iniţial simplificat, model ce permite dezvoltatorilor şi utilizatorilor să testeze şi să amelioreze incremental noua aplicaţie. Un mare avantaj al prototipizării este gradul mult mai mare de acceptare a noului sistem la final, acesta

11. Vezi, spre exemplu, lucrările : [Date 04], [Teorey 99], [Muller 99], [Simsion 01].
12. Vezi, spre exemplu, [Connoly & Begg 02].
13. Preluare din [Connoly & Begg 02], pp. 284-287.

fiind dezvoltat împreună cu utilizatorii ce pot să-şi exprime pe parcurs doleanţele. Etapele prototipizării sunt descrise în figura 1.5[14].

Tabelul 1.1. Criterii de selecţie a SGBD

Definirea datelor	**Definirea fizică**
Gestiunea cheilor primare	Structuri fizice disponibile
Specificarea cheilor străine	Întreţinerea structurii de fişiere
Tipuri de date disponibile	Uşurinţa reorganizării
Extensibilitatea tipurilor de date	Indexare
Specificarea domeniilor	Atribute/înregistrări de mărime variabilă
Uşurinţa restructurării	Compresia datelor
Mecanism de tabele derivate	Rutine de criptare
Controlul integrităţii	Cerinţe de memorie
Dicţionar de date	
Independenţa datelor	**Alte facilităţi/opţiuni**
Tipul de model de date utilizat	Evolutivitate
	Stabilitatea furnizorului
Accesibilitate	Baza de utilizatori
Suport pentru standardele SQL	Pregătire şi suport pentru utilizatori
Interfaţă cu 3GL	Documentare
Multiutilizator	Sistem de operare cerut
Securitate:	Cost
– controlul accesului	*Help online*
– mecanism de autorizare	Standarde utilizate
	Managementul versiunilor
Utilitare	Optimizare a interogărilor
Măsurarea performanţei	Scalabilitate
Acordare (*tuning*)/optimizare	Suport pentru instrumente OLAP
Facilităţi de încărcare/descărcare	Interoperabilitate cu alte SGBD
Suport pentru administrarea BD	Integrare Web
	Utilitare de replicare
Dezvoltare	Facilităţi de distribuire
Instrumente 4GL	Portabilitate
Instrumente CASE	Cerinţe hardware
Facilităţi vizuale	Suport pentru reţea
Proceduri de stocare, declanşatoare	Facilităţi de orientare pe obiect
Instrumente de dezvoltare Web	Arhitecturi pe două, trei... straturi
	Performanţă
Controlul tranzacţiilor	Rata de tranzacţii pe secundă (minut)
Rutine de salvare şi restaurare	Număr maxim de utilizatori simultani
Puncte de salvare intermediare	Suport pentru XML
Suport pentru jurnalizare	
Granularitatea accesului simultan	
Strategia de rezolvare a interblocajelor	
Procesare paralelă a interogărilor	

Implementarea

Implementarea presupune crearea definiţiilor BD la nivelurile conceptual, extern şi intern, precum şi punerea în lucru a programelor (aplicaţiei), fiind realizată utilizând limbajul de definire a datelor (DDL) pus la dispoziţie de SGBD-ul selectat. Implementarea aplicaţiei se realizează într-un mediu de programare utilizând un limbaj de tip 3GL sau 4GL.

14. Preluare din [Connoly ş.a. 96].

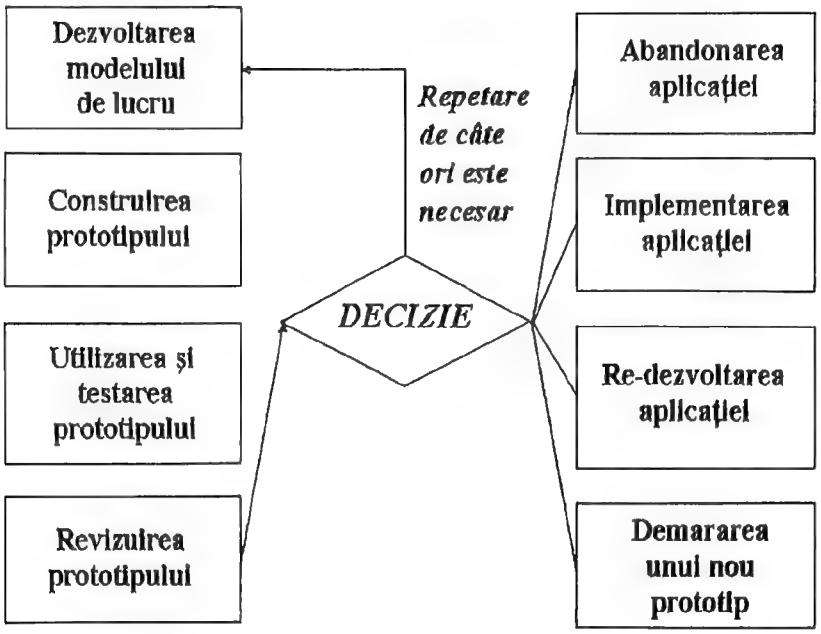


Figura 1.5. Schema de principiu a prototipizării

Conversia şi încărcarea datelor

Acest pas este necesar atunci când sistemul (aplicaţia) dezvoltat înlocuieşte un altul. Datele trebuie transferate din vechea aplicaţie în cea nouă, operaţie ce implică de multe ori schimbarea structurii fişierelor, a formatului, logic şi fizic, al datelor, în concordanţă cu cerinţele noii aplicaţii, dar şi ale noului SGBD. Uneori sunt necesare (şi posibile) şi conversii ale unor programe din vechiul în noul sistem. Majoritatea SGBD-urilor actuale au module de import-export din/în alte SGBD-uri. Conversia aplicaţiilor este, în general, mult mai complexă atunci când se schimbă limbajul/mediul de programare.

Testarea

Testarea este operaţiunea sistematică de verificare a funcţionalităţii sistemului, a gradului de adecvare la cerinţele utilizatorilor. Prin testare, aplicaţia este lansată în execuţie, urmărindu-se depistarea eventualelor erori şi ameliorarea unor parametri precum viteza, securitatea etc. Urmând o metodologie riguroasă, măsurătorile din cadrul testării dau posibilitatea evaluării calităţii şi fiabilităţii software-ului.

Este de preferat ca utilizatorii aplicaţiei să fie pe deplin implicaţi în procesul testării, iar aplicaţia testată să fie instalată pe un alt sistem decât cel pe care a fost concepută; în orice caz, trebuie avut în vedere şi un mecanism de recuperare a datelor corupte în caz de avarie.

Există mai multe strategii de testare a corectitudinii şi funcţionalităţii unei aplicaţii de lucru cu BD ce pot fi aplicate individual sau combinate în cadrul aceluiaşi sistem:

- *top-down*;
- *bottom-up*;
- fire de execuţie;
- test de stres.

Testarea *top-down* începe de la nivelul superior al funcţiilor majore ale aplicaţiei. Este utilă în verificarea arhitecturii sistemului, reproiectările majore fiind semnalate din primele momente ale testării.

Testarea *bottom-up* începe cu modulele elementare şi continuă pe verticală cu modulele compozite până la nivelul general al aplicaţiei.

Testarea *firelor de execuţie* este foarte importantă în sistemele de prelucrare în timp real, în care sunt rulate simultan o serie de procese ce cooperează. Acest tip de testare este mult mai complex, fiind legat de detaliile tehnice ale sistemului de operare. Fiecărui proces i se alocă un fir de execuţie (*thread*) care este urmărit de la declanşarea sa, acordându-se o mare importanţă punctelor de întâlnire cu celelalte fire executate pe sistem.

Testul de *stres* presupune suprasolicitarea bazei şi aplicaţiilor. Astfel, BD se încarcă automat cu un număr extrem de mare de înregistrări, iar la aplicaţie se conectează cât mai mulţi utilizatori. Astfel se depistează până la ce limite rezistă aplicaţia.

Întreţinerea operaţională

Această activitate se derulează pe toată durata de utilizare a aplicaţiei. În afară de monitorizarea BD, a programelor, de „curăţarea" periodică şi repararea eventualelor erori hard/soft, tot în această activitate sunt încorporate operaţiunile de actualizare a datelor şi programelor (ca urmare a unor noi oportunităţi tehnologice), modificarea unor parametri (procente TVA, impozite etc.). Monitorizarea performanţei sistemului se realizează prin raportarea la un nivel prestabilit de acceptare. Dacă este cazul, o situare a performanţei generale sub acest punct critic poate antrena reorganizarea BD. Altminteri, chiar şi în cazul funcţionării în parametri normali, BD trebuie optimizată, având în vedere şi facilităţile oferite de SGBD.

Întreţinerea şi actualizarea aplicaţiei sunt necesare în mai mare sau mai mică măsură. Spre exemplu, într-un mediu economic şi legislativ dinamic, cum este cel din ţara noastră, deseori este necesară modificarea unor părţi importante ale aplicaţiei. Uneori, modificarea presupune reluarea unora sau tuturor paşilor din ciclul de viaţă al aplicaţiei de lucru cu BD.

1.4. Privire de ansamblu asupra proiectării bazelor de date

Deşi au fost publicate destule cărţi dedicate proiectării bazelor de date, subiectul continuă să fie departe de epuizare. Chiar dacă unii autori s-au străduit să aducă rigurozitate, uneori cu preţul unei anume rigidităţi, proiectarea bazelor de date rămâne, după cum spunea şi Chris Date[15], preponderent o artă şi mai puţin o ştiinţă.

Dacă luăm în considerare cele trei niveluri de lucru cu o bază de date din celebra figură 1.1, problema proiectării ţine de elaborarea unei structuri logice şi a alteia

15. [Date 04], p. 330.

fizice ale uneia sau mai multor baze de date în vederea întâmpinării nevoilor informaţionale ale utilizatorilor într-o organizaţie, pentru un set definit de aplicaţii[16].

Una dintre cele mai influente cărţi de proiectare a bazelor de date este cea publicată de Candace Fleming şi Barbara von Halle în 1989[17]. Din păcate, cele două autoare sunt şi întemeietoarele unui întreg şir de inadvertenţe propagate în multe lucrări până în ziua de azi. Spre exemplu, autoarele mărturisesc că materialul lor prezintă două metodologii : *modelarea logică a datelor* şi *proiectarea bazelor de date relaţionale*, definind modelarea logică a datelor ca procedură pentru reprezentarea cerinţelor informaţionale într-un format corect, consistent şi stabil, iar proiectarea BD relaţionale drept procedură de transformare a modelului logic al datelor într-o schemă relaţională stabilă[18].

Cartea de faţă este una dintre dovezile clare că modelul logic al datelor, mai precis, structura conceptuală a bazei de date poate fi concepută şi strict în termenii modelului relaţional. Greşeala autoarelor ţine de faptul că, pentru ele, o schemă relaţională alcătuită din tabele şi restricţiile la care sunt supuse datele din tabele nu ar corespunde nivelului logic (conceptual) al bazei de date, ci mai degrabă nivelului fizic. După Fleming şi von Halle, metodologia de *modelare logică a datelor* (MLD) se derulează în 12 paşi[19] :

1. identificarea entităţilor majore ;
2. determinarea relaţiilor dintre entităţi ;
3. determinarea cheilor primare şi alternative ;
4. determinarea cheilor străine ;
5. determinarea celor mai importante reguli organizaţionale (*key business rules*) ;
6. adăugarea celorlalte atribute (atributele non-cheie) ;
7. validarea perspectivelor utilizatorului folosind normalizarea ;
8. determinarea domeniilor (restricţiilor asupra valorilor autorizate ale atributelor) ;
9. determinarea operaţiunilor declanşatoare (regulile care guvernează efectele modificărilor asupra atributelor şi entităţilor) ;
10. combinarea perspectivelor utilizatorului ;
11. integrarea cu modelele de date existente ;
12. analiza stabilităţii şi dezvoltărilor ulterioare.

Este evident că, deşi autoarele separă artificial proiectarea logică a bazei de date de proiectarea bazei de date relaţionale, o parte dintre paşii modelării logice sunt raportaţi explicit la modelul relaţional. Perspectiva poate fi explicată prin faptul că, la nivelul anului 1989, modelul suveran în materie de gestiune a bazelor de date era cel relaţional (suveranitate manifestată din plin şi astăzi), iar SGBD-urile ce puteau exploata baze de date obiectuale erau ca şi inexistente.

16. [Elmasri & Navathe 00], p. 532.
17. [Fleming & von Halle 89].
18. [Fleming & von Halle 89], p. 6.
19. [Fleming & von Halle 89], pp. 19-25.

Pentru a vă face o idee completă asupra părţii a doua din proiectarea bazelor de date în viziunea celor două autoare, iată o scurtă enumerare a principalelor etape din aşa-numita *proiectare a bazei de date relaţionale* (PBDR) care continuă operaţiunile derulate în MLD[20]:

1. identificarea tabelelor;
2. identificarea atributelor;
3. adaptarea structurii datelor la specificul SGBD-ului folosit;
4. inventarierea regulilor organizaţionale privind entităţile;
5. inventarierea regulilor organizaţionale privind relaţiile (asociaţiile) dintre entităţi;
6. inventarierea regulilor adiţionale despre atribute;
7. optimizarea structurii în vederea unui mecanism de acces cât mai eficient;
8. definirea secvenţelor de „înmănunchiere" (*clustering*);
9. definirea cheilor pentru calcularea adreselor logice ale înregistrărilor în tabele (*hash keys*);
10. adăugarea indexurilor;
11. adăugarea de date redundante;
12. redefinirea coloanelor;
13. redefinirea tabelelor.

Literatura de specialitate şi practica se raportează însă preponderent la doar două componente majore ale proiectării bazelor de date: proiectarea *logică* şi proiectarea *fizică*. Din acest punct de vedere, cele 12 etape din MLD şi primele 6 din aşa-numita PBDR ar corespunde, în linii mari, proiectării logice a bazei, în timp ce etapele următoare (7-13) din PBDR – proiectării fizice.

O altă abordare, prezentată în [Teorey 99], structurează ciclul de viaţă al bazelor de date pe următoarele etape[21]:

1. analiza cerinţelor;
2. proiectarea logică, ce se materializează în schema globală ce conţine datele şi relaţiile dintre ele, fiind derulată astfel:
   a) modelarea E-R;
   b) integrarea perspectivelor utilizator;
   c) conversia diagramelor E-R în tabele[22];
   d) normalizarea tabelelor;
3. proiectarea fizică: selectarea indexurilor, metodelor de acces, *cluster*-elor de date; tot aici, Teorey include şi denormalizarea;
4. distribuirea datelor, materializată în schema de fragmentare şi schema de alocare a datelor;
5. implementarea, monitorizarea şi modificarea ulterioară a bazei de date. Şi aici apar dubii legate de modul de derulare a normalizării (2.d), o dată ce prin conversia diagramelor E-R se obţin deja tabelele relaţionale (2.c).

20. [Fleming & von Halle 89], pp. 72-79.
21. [Teorey 99], pp. 4-10.
22. Teorey le numeşte *tabele SQL*.

Deşi clasice, ambele lucrări din care am extras activităţile de mai sus sunt legate îndeosebi de paradigma relaţională, folosind, ce-i drept, în primele etape modelul E-R, mai apropiat de lumea reală. Robert Muller abordează în cartea sa atât aspectele generale ale proiectării, cât şi pe cele specifice fiecăruia dintre cele trei modele în vogă astăzi: relaţional, obiectual şi relaţional-obiectual. Pentru autor, procesul proiectării bazelor de date este prin excelenţă unul iterativ, incremental, într-o măsură cu atât mai mare cu cât ne apropiem de obiectual. Fiecare iteraţie se materializează într-o formă a schemei bazei, pornindu-se de la modelare, apoi trecându-se de la proiectare la construcţie (implementare)[23]. Cu toate acestea, activităţile principale ale ciclului de viaţă al bazei de date nu diferă prea mult, cel puţin la nivel general[24]:

- analiza cerinţelor informaţionale;
- modelarea datelor;
- proiectarea şi optimizarea bazei de date (proiectare logică şi fizică);
- testarea şi revizuirea bazei;
- certificarea bazei de date;
- întreţinerea şi ameliorarea bazei.

Specifică lucrării lui Muller este activitatea de certificare a bazei care se constituie ca un fel de atestat de bună funcţionare acordat bazei, atestat menit a întări încrederea beneficiarilor bazei.

Connoly şi Begg[25] definesc însă trei tipuri: *proiectarea conceptuală*, ce ţine de construirea unui model informaţional independent de orice considerent privind aspectul fizic al datelor; *proiectarea logică*, ce constă în construirea unui model informaţional pe calapodul unui model de date consacrat (E-R, relaţional, OO, OR), dar independent de tipul SGBD-ului ales şi de celelalte aspecte fizice ale modelului; *proiectarea fizică*, ce se materializează în implementarea efectivă a bazei pe suportul de stocare, inclusiv aspecte ce ţin de asigurarea unui acces eficient şi a securităţii.

Iar ca lucrurile să se complice şi mai rău, se cuvine de remarcat că Elmasri şi Navathe delimitează, precum Connoly şi Begg, trei categorii ale proiectării, însă pentru ei proiectarea logică constă în conversia schemei conceptuale într-un model de date *dependent* de SGBD-ul care va fi folosit[26].

Fideli majorităţii, vom alege varianta mai simplă, cu cele două tipuri de proiectare, logică şi fizică, deşi în capitolele următoare ne vom raporta, aproape exclusiv, la primul.

23. [Muller 99], p. 1.
24. [Muller 99], pp. 1-10.
25. [Connoly & Begg 02], pp. 281-282.
26. [Elmasri & Navathe 00], pp. 532-535.

1.5. Proiectarea bazelor de date în prezentul volum

Materialul prezentat până în acest moment este destinat a contura arealul acoperit de restul acestei lucrări. Mai precis, acest prim volum dedicat proiectării bazelor de date se „remarcă" prin deraierea de la metodologia ortodoxă de proiectare a bazelor de date. Practic, nu vom discuta nici de modelul entităţi-relaţii, nici de diagrame UML, nici de perspective utilizatorului. Este posibilă proiectarea bazelor de date fără a avea deprinderi elementare în diagrame E-R, UML etc. ? Contrar majorităţii covârşitoare a autorilor, eu spun că da!

Aceasta nu înseamnă că tot ce ţine de analiza şi proiectarea sistemelor informaţionale reprezintă un balast inutil, ci doar că, în foarte multe situaţii, o bază de date poate fi adusă la o formă rezonabilă urmând metodologia descrisă în următoarele capitole. Dacă vreţi, este o repunere în drepturi, destul de tardivă însă, a metodologiei „curat relaţionale" într-o perioadă în care cuvintele-cheie în analiză şi proiectare sunt E-R, OO, Rational Rose, UML etc. Abia după ce vom fi epuizat discuţia despre suişurile şi coborâşurile metodologiei bazate pe normalizare şi postnormalizare, vom reveni în „lumea bună" a proiectării, în speratul volum al doilea dedicat proiectării bazelor de date.

De fapt, şi celelalte câteva cărţi publicate la noi în care se abordează problematica proiectării bazelor de date sunt destul de eterogene ca metodologie. Spre exemplu, Octavian Bâscă tratează problema în capitolul 5 al cărţii sale din 1997[27], însă, pe de o parte, modalitatea de tratare este copleşitor teoretică, exemplele fiind cu totul fragmentare, iar, pe de altă parte, în capitol sunt incluse şi chestiuni mai puţin legate de construirea unei baze de date, cum ar fi optimizări ale interogărilor.

Metodologia prezentată în Lungu ş.a.[28] este ceva mai laborioasă, fără însă a rivaliza cu cea din lucrările clasice străine, bazându-se pe patru etape:

- analiza sistemului economic şi a cerinţelor informaţionale asociate;
- proiectarea structurilor conceptuale, externe şi interne ale bazei de date;
- încărcarea datelor în bază;
- exploatarea şi întreţinerea bazei de date.

Metodologia foloseşte modelul E-R, însă, curios lucru, diagramele E-R sunt incluse în etapa de analiză a sistemului, nu în cea de proiectare a structurii. De asemenea, este discutabilă desemnarea activităţii de alegere a sistemului de gestiune a bazei de date drept primă etapă în proiectarea structurii, precum şi modul în care autorii delimitează faza de încărcare a datelor de cea de exploatare şi întreţinere a bazei.

Probabil singura carte românească dedicată, cel puţin explicit, exclusiv proiectării bazelor de date este cea a dnei Ileana Popescu[29]. Trecând peste amănuntul includerii

27. [Bâscă 97], pp. 90-155.
28. [Lungu ş.a. 95], pp. 26-59.
29. [Popescu 01].

arhitecturii interne a sistemului Oracle într-o lucrare de modelare (pp. 26-37), lucrării i se pot aduce şi laude, şi comentarii (răutăcioase), cum ar fi:

- în capitolul 2, dedicat proiectării bazelor de date relaţionale, mai întâi se transformă diagramele E-R în tabele relaţionale (paragraful 2.3), iar abia apoi se prezintă normalizarea, fără a se înţelege ce anume se normalizează, atâta vreme cât schema bazei a fost deja obţinută;
- este discutabilă includerea în capitolul 2 a limbajelor de interogare, de fapt, concentrarea exclusivă pe aspectul manipulativ al lor (DML), şi nu pe cel de definire a datelor (DDL);
- regulile lui Codd (paragraful 2.5) sunt reguli ale unui SGBD relaţional, nu ale proiectării unei baze de date relaţionale;
- prezentarea utilitarului SQL*Plus (paragraful 2.8.2) şi a limbajului de programare (extensia procedurală a SQL-ului) Oracle PL/SQL (paragraful 2.8.3) în subcapitolul dedicat normalizării este întru totul discutabilă.

În concluzie, cărţilor româneşti dedicate proiectării bazelor de date li se poate imputa un cert „defect metodologic", dar mai ales inapetenţa cronică pentru aplicativ, lipsa oricăror soluţii destinate practicienilor. Nu este de mirare faptul că, atât la noi, cât şi în majoritatea lucrărilor internaţionale, bazele de date se află în postura unui conglomerat din care fiecare înţelege ce vrea şi ce poate.

Pornind de la faptul că, în esenţă, un model de date este legat de trei aspecte, *structură* (modelul de organizare a datelor), *integritate* (restricţiile ce pot fi definite asupra datelor pentru a asigura corectitudinea informaţiilor din bază) şi *manipulare* (modalităţi de editare şi extragere a informaţiilor din bază), capitolele ce urmează sunt concentrate pe aspectul structural şi pe mecanismul de integritate ale unei baze de date. Proiectarea fizică, mecanismele de control, concurenţă şi parametrii fizici ai bazei de date sunt, în cea mai mare parte, eludate în acest volum.

Practic, lucrarea de faţă îşi propune să vă supună atenţiei cum anume s-ar putea „gândi" o schemă de bază de date pentru o aplicaţie anume, ce elemente ar trebui luate în considerare, chestiunile mai riguroase şi cele mai laxe. Întregul demers este bazat pe normalizare şi pe operaţiuni de postnormalizare. În plus, ca parte integrantă a bazei de date, un spaţiu generos este acordat mecanismului de declarare a restricţiilor pentru asigurarea într-o cât mai mare măsură a integrităţii, coerenţei şi consistenţei datelor din bază. Din acest punct de vedere, vom urma observaţia lui Chris Date, după care proiectarea BD nu este doar o chestiune de găsire a celor mai bune structuri de stocare; integritatea datelor nu reprezintă doar un aspect, ci aspectul-cheie[30].

30. [Date 04], p. 330.

CAPITOLUL 2

La început a fost normalizarea

Moto :

Se poate elabora un întreg capitol sau chiar o carte despre normalizare, însă este greu de scris ceva uşor şi plăcut de citit. (Rober J. Muller[1])

O carte dedicată proiectării bazelor de date ce începe abrupt cu normalizarea şi se termină aproape la fel poate naşte discuţii îndreptăţite. După cum am văzut şi în primul capitol, multe lucrări conturează la început una sau mai multe metodologii de proiectare a bazelor de date, iar normalizarea este inclusă undeva spre miez, ca etapă, distinctă sau nu, pentru obţinerea schemei finale a bazei de date[2]. Pe de altă parte, normalizarea este cât se poate de tehnică, ba chiar ştiinţifică pe alocuri, ceea ce poate îndepărta cititorii mai puţin răbdători de la lectura următoarelor capitole.

De ce totuşi ne încăpăţânăm să încălcăm o reţetă de (relativ) succes şi consacrăm acest prim volum dependenţelor dintre atribute şi formelor normalizate ale unei baze de date relaţionale? Mai întâi, din raţiuni istorico-sentimentale. Edgar F. Codd, Dumnezeu să-l odihnească – s-a prăpădit în primăvara lui 2003 –, a fundamentat modelul relaţional de organizare a datelor împreună cu primele trei forme normale[3]. Continuată prin munca multor specialişti, normalizarea a fost prima teorie închegată de elaborare a schemei unei baze de date.

Este drept că astăzi sunt mai la îndemână modele precum E-R (entităţi-relaţii sau entităţi-asociaţii), mai lizibile, expresive şi uşor de „mânuit", însă acestea apar mai târziu. Cel la care facem trimitere, modelul *entităţi-relaţii*, a fost elaborat de Peter P. Chen pe la mijlocul anilor '70, iar altele pe parcursul anilor '80 şi '90.

În al doilea rând, dacă modelele ce-şi au sorgintea în E-R, OO, ORM etc. au creat un veritabil turn Babel al formalismelor şi notaţiilor, fiecare variantă având deopotrivă suporteri şi detractori la fel de înverşunaţi, normalizarea are o imagine de standard, metodologie unică, riguroasă (este drept, cam „scorţoasă" pe alocuri). Vom vedea,

1. [Muller 99], p. 321.
2. [Harrington 02], [Fleming & von Halle 89], [Riordan 99], [Teorey 99], [Hernandez 97], [O'Neil & O'Neil 01], [Elmasri & Navathe 00].
3. După cum preciza şi Chris Date [Date 86], p. 364, normalizarea nu este considerată parte a modelului relaţional, ci o teorie separată construită „deasupra" modelului.

însă, nu peste mult timp, că şi în privinţa normalizării lucrurile nu sunt definitive, ba, mai mult, excesul de normalizare strică de multe ori schema bazei de date.

Nu în ultimul rând, pentru aplicaţiile relativ simple (chiar medii) şi stabile în timp, normalizarea rămâne un mijloc la îndemână de a „croi" schema unei baze de date. Dacă lucrurile se complică însă, este necesar să se recurgă la alte metodologii de proiectare.

Cum normalizarea este legată ombilical de modelul relaţional, primul paragraf îl vom cheltui cu parcurgerea sumară a noţiunilor fundamentale ale relaţionalului.

2.1. Concepte ale modelului relaţional de organizare a bazelor de date

Descrierea unui model de date se bazează, în principal, pe trei elemente: *structura* – modul în care sunt organizate efectiv datele în bază, *integritatea* – mecanismul de asigurare a corectitudinii datelor, şi *manipularea* – în sensul modificării şi extragerii informaţiilor din bază. După cum spuneam şi în capitolul anterior, pentru această lucrare ne interesează primele două aspecte, deşi, vrând-nevrând, atunci când ne vom pune problema implementării tuturor restricţiilor bazei în oricare dintre SGBD-uri, vom recurge copios la opţiunile manipulative ale modelului.

Modelul relaţional s-a conturat în două articole publicate în 1969 şi 1970 de către E.F. Codd, matematician la centrul de cercetări din San Jose (California) al firmei IBM. În acel moment, tehnologia bazelor de date era centrată pe modelele ierarhic şi reţea, modele ce depind într-o mai mare măsură de organizarea internă a datelor pe suportul de stocare. Codd a propus o structură de date tabelară, independentă de tipul de echipamente şi de software-ul de sistem pe care este implementată, structură „înzestrată" cu o serie de operatori pentru extragerea datelor. Deşi puternic matematizat, modelul relaţional este relativ uşor de înţeles, fiind definit printr-o serie de structuri de date (relaţii alcătuite din tupluri), operaţii aplicate asupra structurilor de date (selecţie, proiecţie, joncţiune etc.) şi reguli de integritate care să asigure consistenţa datelor (chei primare, restricţii referenţiale ş.a.).

Faţă de modelele ierarhic şi reţea, modelul relaţional prezintă câteva avantaje:

- propune structuri de date uşor de utilizat;
- ameliorează independenţa logică şi fizică;
- pune la dispoziţia utilizatorilor limbaje neprocedurale;
- optimizează accesul la date;
- îmbunătăţeşte integritatea şi confidenţialitatea datelor;
- ia în calcul o largă varietate de aplicaţii;
- abordează metodologic definirea structurii bazei de date.

Dacă, teoretic, modelul s-a consacrat în anii '70, produsele software care să gestioneze baze de date relaţionale au devenit populare abia în deceniul 8. Deşi IBM a fost prima firmă care a iniţiat un proiect destinat elaborării unui SGBD relaţional (System/R, începând cu 1974), prima firmă care a lansat un SGBDR comercial a fost Relational

Software, Inc., astăzi Oracle. Înfiinţarea firmei a avut loc în 1977, iar lansarea produsului în 1979. Impunerea pe piaţă a SGBD-urilor grefate pe modelul relaţional a fost mult mai dificilă decât s-ar putea înţelege astăzi. Abia în deceniul 9, datorită vitezei ameliorate, securităţii sporite şi mai ales productivităţii dezvoltatorilor de aplicaţii, modelul relaţional a câştigat supremaţia.

Există o largă tipologie a SGBDR-urilor, aşa încât orice categorisire îşi are riscul său. Cele mai accesibile şi, implicit, cele mai utilizate sunt SGBDR-urile dedicate iniţial uzului individual: Access, Paradox, Visual FoxPro. Astăzi, multe dintre acestea au caracteristici profesionale şi pot fi folosite simultan la dezvoltarea aplicaţiilor pentru zeci de utilizatori. Pentru aplicaţiile complexe din bănci, corporaţii, organizaţii şi instituţii de mari dimensiuni s-au impus SGBDR-urile de „categoria grea": Oracle, DB2 (IBM), Informix (IBM), Sybase (Sybase), SQL Server (Microsoft). Acestea sunt mult mai robuste, mult mai fiabile, dar şi costisitoare.

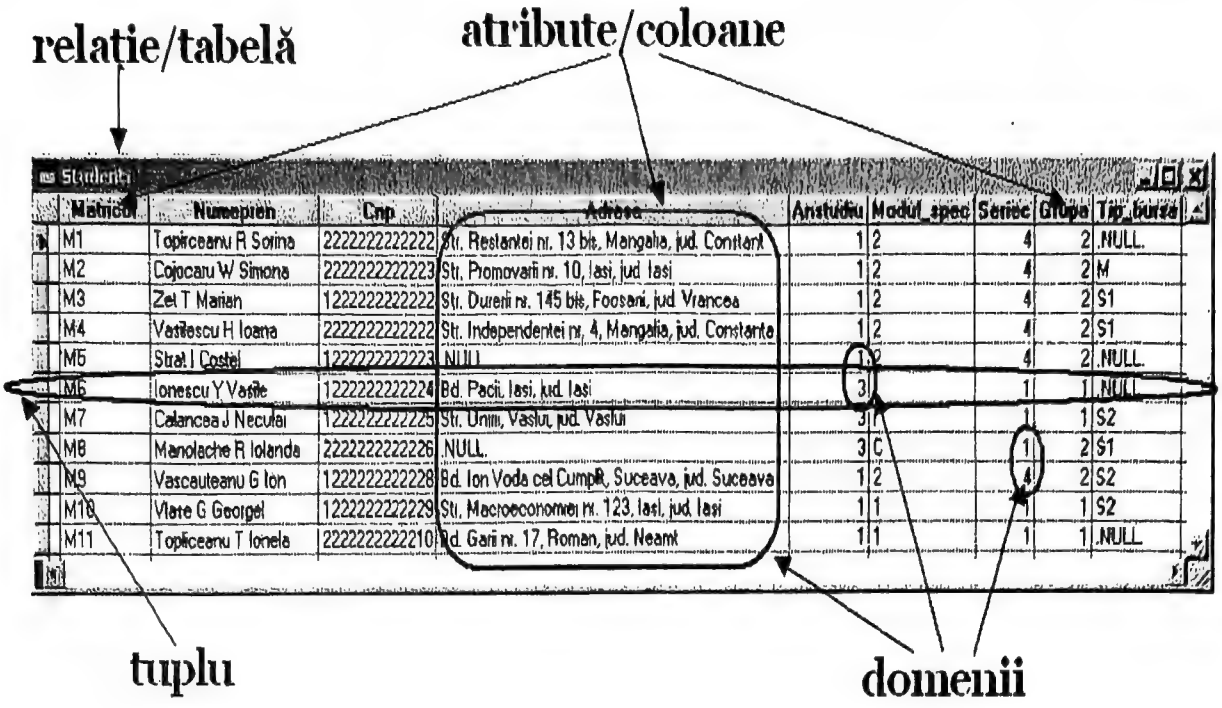
În ultimul timp şi-au făcut apariţia, ca alternative la sus-amintiţii coloşi, aşa-zisele SGBD-uri (aproape) gratuite, precum PostgreSQL, MySQL, mSQL, Firebird etc. Acestea rulează, de obicei, pe sisteme de operare de tip Linux (foarte ieftine) şi se întrevăd ca adversari serioşi ai marilor producători.

2.1.1. *Relaţii/tabele, domenii, atribute, valori nule*

La modul simplist, o bază de date relaţională (BDR) poate fi definită ca un ansamblu de relaţii (tabele); fiecare tabelă (sau tabel), alcătuită din linii (tupluri), are un nume unic şi este stocată pe suport extern (de obicei disc). La intersecţia unei linii cu o coloană se găseşte o valoare atomică. O relaţie conţine informaţii omogene legate de anumite entităţi, procese, fenomene: CĂRŢI, ELEVI, LOCALITĂŢI, PERSONAL, FACTURI etc. Spre exemplu, în figura 2.1 este reprezentată tabela STUDENŢI ce stochează informaţii privitoare la studenţii înscrişi la cursurile unei facultăţi.

În teoria relaţională se foloseşte termenul *relaţie*. Practica însă a consacrat termenul *tabelă* (engl. *table*). Un *tuplu* sau o *linie* este o succesiune de valori de diferite tipuri. În general, o linie grupează informaţii referitoare la un obiect, eveniment etc., altfel spus, informaţii referitoare la o entitate: o carte (un titlu sau un exemplar din depozitul unei biblioteci, depinde de circumstanţe), un/o student(ă), o localitate (oraş sau comună), un angajat al firmei, o factură emisă etc. În figură este pus în evidenţă al şaselea tuplu din tabela STUDENŢI, tuplu referitor la studentul *Ionescu Y. Vasile*. Linia respectivă este alcătuită din nouă valori ce desemnează: matricolul studentului; numele, iniţiala tatălui şi prenumele; codul numeric personal (CNP-ul), adresa (stabilă); anul de studiu; modulul (dacă e în anul 1 sau 2) sau specializarea (dacă e în anii 3, 4 sau 5) la care este înscris studentul; seria de curs; grupa; tipul de bursă de care beneficiază.

Teoretic, orice tuplu reprezintă o relaţie între clase de valori (în cazul nostru, între nouă clase de valori); de aici provine sintagma *baze de date relaţionale*, în sensul matematic al relaţiei, de asociere a două sau mai multe elemente. Firește, toate tuplurile relaţiei au acelaşi format (structură). Ordinea tuplurilor nu prezintă importanţă din punctul de vedere al conţinutului informaţional al tabelei.



Matricol	Numeprien	Cnp	Adresa	Anstudiu	Modul_spec	Seriec	Grupa	Tip_bursa
M1	Topliceanu R Sorina	2222222222222	Str. Restaurantei nr. 13 bis, Mangalia, jud. Constanta	1	2	4	2	.NULL.
M2	Cojocaru W Simona	2222222222223	Str. Promovarii nr. 10, Iasi, jud. Iasi	1	2	4	2	M
M3	Zet T Marian	1222222222222	Str. Durerii nr. 145 bis, Focsani, jud. Vrancea	1	2	4	2	S1
M4	Vasilescu H Ioana	2222222222222	Str. Independentei nr. 4, Mangalia, jud. Constanta	1	2	4	2	S1
M5	Stratl Costel	1222222222223	NULL	1	2	4	2	.NULL.
M6	Ionescu Y Vasile	1222222222224	Bd. Pacii, Iasi, jud. Iasi	3		1	1	.NULL.
M7	Calancea J Neculai	1222222222225	Str. Unirii, Vaslui, jud. Vaslui	3		1	1	S2
M8	Manolache R Iolanda	2222222222226	.NULL.	3	C	1	2	S1
M9	Vascauteanu G Ion	1222222222228	Bd. Ion Voda cel Cumpt, Suceava, jud. Suceava	1	2	4	2	S2
M10	Vlase G Georgel	1222222222229	Str. Macroeconomei nr. 123, Iasi, jud. Iasi	1	1	1	1	S2
M11	Topliceanu T Ionela	2222222222210	Bd. Gari nr. 17, Roman, jud. Neamt	1	1	1	1	.NULL.

Figura 2.1. Relația (tabela) STUDENȚI

Fiecare atribut este caracterizat printr-un nume şi un domeniu de valori pe care le poate lua. *Domeniul poate fi definit ca ansamblul valorilor acceptate (autorizate) pentru un element component al relației*:

- într-o tabelă destinată datelor generale ale angajaților, pentru atributul `Sex`, domeniul este alcătuit din două valori: *femeiesc* şi *bărbătesc*;
- domeniul atributului `Judet` este alcătuit din numele fiecărui judeţ (plus Bucureşti);
- domeniul unui atribut precum `PretUnitar`, care se referă la preţul la care a fost vândut un produs/serviciu, este cu mult mai larg, fiind alcătuit din orice valoare cuprinsă între 1 şi 99999999999 lei (având în vedere rata inflației).

Reţinem corespondenţa noţiunilor relaţie-tabelă, tuplu-linie şi atribut-coloană. Numărul de tabele pe care le conţine o bază de date, atributele „adunate" în fiecare tabelă, domeniul fiecăruia dintre atribute prezintă diferenţe majore de la o bază la alta, chiar dacă uneori reflectă acelaşi tip de procese. Intrăm astfel în sfera proiectării bazelor de date, a dependenţelor şi normalizării.

În figura 2.1 se observă că atributul `Adresa` conţine, pe linia (tuplul) 8, o valoare curioasă, notată NULL. Valoarea NULL este considerată o *metavaloare* şi indică faptul că, în acel loc, informaţia este necunoscută sau inaplicabilă. În cazul nostru, studentei *Manolache R. Iolanda* nu i se cunoaşte adresa. Valoarea NULL este diferită de valorile 0 sau spaţiu.

În încheiere, principalele caracteristici ale unei relaţii sunt sistematizate după cum urmează:

- în cadrul unei baze de date, o relaţie prezintă un nume distinct de al celorlalte relaţii;

- valoarea unui atribut într-un tuplu oarecare conţine a singură valoare (o valoare atomică);
- fiecare atribut are un nume distinct;
- orice valoare a unui atribut face parte din domeniul pe care a fost definit acesta;
- ordinea dispunerii atributelor în relaţie nu prezintă importanţă;
- fiecare tuplu este distinct, adică nu pot exista două tupluri identice;
- ordinea tuplurilor nu influenţează conţinutul informaţional al relaţiei.

2.1.2. *Restricţii ale bazei de date*

Principalele restricţii definibile în modelul relaţional sunt: de domeniu, de nenulitate, de atomicitate, de unicitate, referenţială şi cele definite de utilizator.

Restricţia de domeniu

După cum am văzut în paragraful anterior, un atribut este definit printr-un nume şi un domeniu. Orice valoare a atributului trebuie să se încadreze în domeniul definit. Există mai multe moduri de percepţie a acestei restricţii. O parte din informaticieni substituie domeniul tipului atributului: numeric, şir de caractere, dată calendaristică, logic (boolean) etc., şi, eventual, lungimii (numărul maxim de poziţii/caractere pe care se poate „întinde" un atribut). După cum se observă, este luat în calcul numai aspectul sintactic al domeniului. Faptul că anul de studiu al unei clase poate fi una dintre valorile 9, 10, 11, 12 reprezintă o restricţie de comportament sau, mai simplu, o restricţie definită de utilizator.

Cea de-a doua categorie priveşte deopotrivă domeniile sintactic şi semantic. Astfel, domeniul sintactic al atributului `LitClasa` (litera) dintr-o relaţie precum ELEVI_LICEU este un caracter, obligatoriu literă, şi chiar mai restrictiv, litera e obligatoriu majusculă. Din punct de vedere semantic, `LitClasa` poate lua una dintre valorile: A, B, C, ..., în funcţie de numărul de clase dintr-un an de studiu (cinci clase de a IX-a, patru de a X-a etc.).

Majoritatea SGBD-urilor permit definirea tuturor elementelor ce caracterizează domeniul (sintactic şi semantic) atributelor prin declararea tipului şi lungimii atributului şi prin aşa-numitele reguli de validare la nivel de câmp (*field validation rules*). Sunt însă şi produse la care domeniul poate fi definit explicit, sintactic şi semantic, dându-i-se un nume la care vor fi legate atributele în momentul creării tabelelor.

Nenulitatea

Deşi inventată de însuşi E.F. Codd, valoarea nulă este vehement contestată de cea mai mare parte a liniei purist-relaţionale (Date, Darwen, Pascal). Nu discutăm aici legitimitatea unei asemenea meta-valori, ci doar amintim că, într-o bază de date, unora dintre atribute li se poate interzice valoarea NULL.

Atomicitate

Conform teoriei bazelor de date relaţionale, orice atribut al unei tabele oarecare trebuie să fie atomic, în sensul imposibilităţii descompunerii sale în alte atribute.

În aceste condiţii, se spune că baza de date se află în prima formă normală sau prima formă normalizată (1NF). În tabela STUDENŢI un atribut cu valori neatomice poate fi considerat, pentru majoritatea situaţiilor, `Adresa`, întrucât valorile sale îngrămădesc elemente ce pot fi şi individualizate: strada, număr, bloc, scara, etaj, apartament, localitate şi judeţ.

Astăzi, atomicitatea valorilor atributelor a devenit o ţintă predilectă a „atacurilor duşmănoase" la adresa modelului relaţional, datorită imposibilităţii înglobării unor structuri de date mai complexe, specifice unor domenii ca: proiectare asistată de calculator, baze de date multimedia etc. Pentru detalii privind atomicitatea, vezi capitolul următor.

Unicitate

Într-o relaţie nu pot exista două linii identice (două linii care prezintă aceleaşi valori pentru toate atributele). Mai mult, majoritatea relaţiilor prezintă un atribut sau o combinaţie de atribute care diferenţiază cu siguranţă un tuplu de toate celelalte tupluri ale relaţiei. Cheia primară a unei relaţii (tabele) este un atribut sau un grup de atribute care identifică fără ambiguitate fiecare tuplu (linie) al relaţiei (tabelei). Există trei restricţii pe care trebuie să le verifice cheia primară:

- unicitate: o cheie identifică un singur tuplu (linie) al relaţiei;
- compoziţie minimală: atunci când cheia primară este compusă, nici un atribut din cheie nu poate fi eliminat fără distrugerea unicităţii tuplului în cadrul relaţiei; în cazuri limită, o cheie poate fi alcătuită din toate atributele relaţiei;
- valori nenule: valorile atributului (sau ale ansamblului de atribute) ce desemnează cheia primară sunt întotdeauna specificate, deci nenule; mai mult, nici un atribut din compoziţia cheii primare nu poate avea valori nule. Cea de-a treia condiţie se mai numeşte şi *restricţie a entităţii*.

Domeniul unui atribut care este cheie primară într-o relaţie este denumit domeniu primar. Dacă într-o relaţie există mai multe combinaţii de atribute care conferă unicitate tuplului, acestea sunt denumite chei candidate. O cheie candidată care nu este identificator primar este referită ca cheie alternativă. În tabela STUDENŢI există două chei candidate, `Matricol` şi `CNP`. Pe baza unor criterii tainice (deocamdată), am ales `Matricol` drept cheie primară, `CNP`-ul devenind cheie alternativă.

Dacă cheia primară a tabelei STUDENŢI este una simplă, există însă suficiente cazuri în care cheia primară este compusă din două, trei ş.a.m.d. sau, la extrem, toate atributele relaţiei. Să luăm spre analiză o relaţie PERSONAL care conţine date generale despre angajaţii firmei. Fiecare tuplu al relaţiei se referă la un angajat, atributele fiind: `Nume`, `Prenume`, `DataNasterii`, `Vechime`, `SalariuTarifar`.

- Atributul `Nume` nu poate fi cheie, deoarece chiar şi într-o întreprindere de talie mijlocie este posibil să existe doi angajaţi cu acelaşi nume.
- Dacă apariţia a două persoane cu nume identice este posibilă, atunci apariţia a două persoane cu acelaşi `Prenume` este probabilă.
- Nici unul dintre atributele `DataNasterii`, `Vechime` şi `SalariuTarifar` nu poate fi „înzestrat" cu funcţia de identificator.

- În acest caz, se încearcă gruparea a două, trei, patru ş.a.m.d. atribute, până când se obţine combinaţia care va permite diferenţierea clară a oricărei linii de toate celelalte.
- Combinaţia `Nume+Adresă` pare, la prima vedere, a îndeplini „cerinţele" de identificator. Ar fi totuşi o problemă: dacă în aceeaşi firmă (organizaţie) lucrează împreună soţul şi soţia? Ambii au, de obicei, acelaşi nume de familie şi, tot de obicei, acelaşi domiciliu. Este adevărat, cazul ales nu este prea fericit. Dar este suficient pentru compromiterea combinaţiei.
- Următoarea tentativă este grupul `Nume+Prenume+Adresă`, combinaţie neoperantă dacă în organizaţie lucrează tatăl şi un fiu (sau mama şi o fiică), având aceleaşi nume şi prenume şi domiciliul comun.
- Ar rămâne de ales una dintre soluţiile (`Nume+Prenume+Adresă+Vechime`) sau (`Nume+Prenume+Adresa+DataNasterii`).

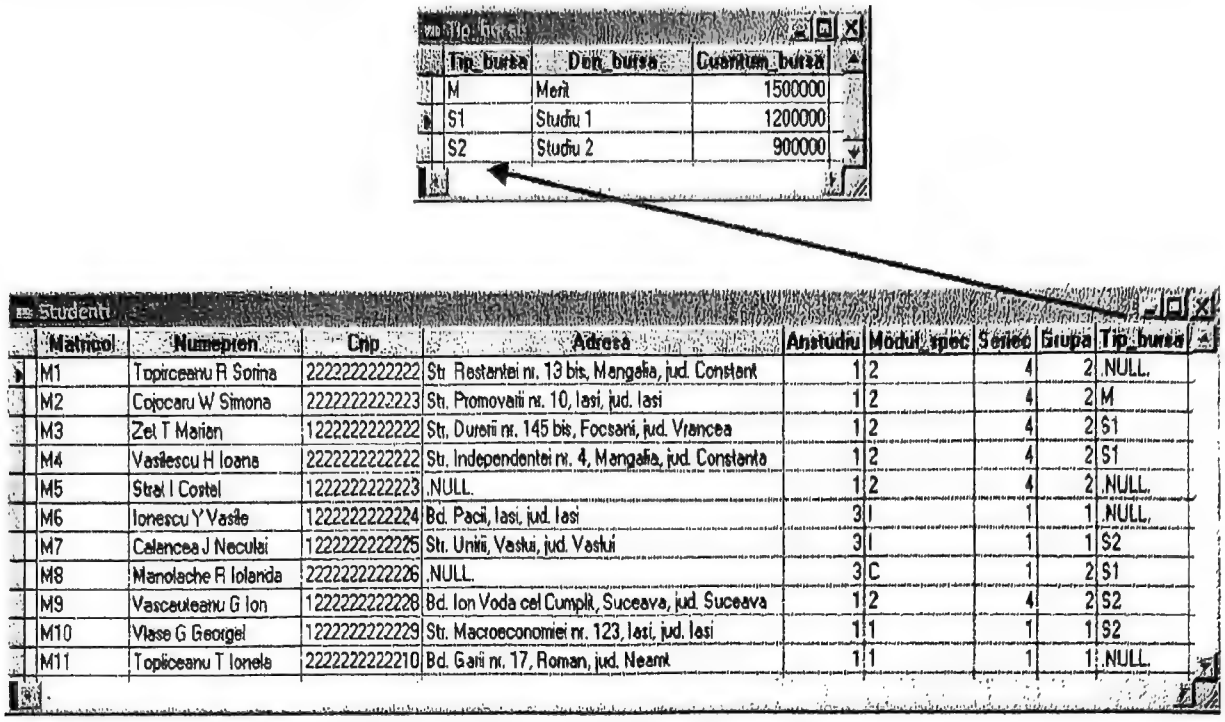
Oricare dintre cele două combinaţii prezintă riscul violării restricţiei de entitate, deoarece este posibil ca, la preluarea unui angajat în bază, să nu i se cunoască adresa sau data naşterii, caz în care atributul respectiv ar avea valoarea NULL. Dificultăţile de identificare fără ambiguitate a angajaţilor au determinat firmele ca, la angajare, să aloce fiecărei persoane un număr unic, număr denumit `Marca`. Prin adăugarea acestui atribut la cele existente, pentru relaţia PERSONAL, problema cheii primare este rezolvată mult mai simplu. Actualmente, sarcina este simplificată şi prin utilizarea codului numeric personal (`CNP`), combinaţie de 13 cifre care prezintă avantajul că rămâne neschimbată pe tot parcursul vieţii persoanei.

Restricţia referenţială

O bază de date relaţională este alcătuită din relaţii (tabele) aflate în legătură. Stabilirea legăturii se bazează pe mecanismul cheii străine şi, implicit, al restricţiei referenţiale. Figura 2.2 prezintă o relaţie în care sunt implicate tabelele TIP_BURSE şi STUDENŢI. Atributul `Tip_Bursa` joacă un rol de „agent de legătură" între cele două relaţii. Pentru tabela TIP_BURSE, atributul `Tip_Bursa` este cheie primară, în timp ce în tabela STUDENŢI, `Tip_Bursa` reprezintă o *coloană de referinţă* sau *cheie străină* (externă), deoarece numai pe baza valorilor sale se poate face legătura cu relaţia-părinte TIP_BURSE. Cheile străine sau coloanele de referinţă sunt deci atribute sau combinaţii de atribute care pun în legătură linii (tupluri) din relaţii diferite. Tabela în care atributul de legătură este primar se numeşte tabelă-părinte (în cazul nostru, TIP_BURSE), iar cealaltă, tabelă-copil.

Legat de noţiunea de cheie străină apare conceptul de *restricţie referenţială*. O restricţie de integritate referenţială apare atunci când o relaţie face referinţă la o altă relaţie. Când două tabele (relaţii), T1 şi T2, prezintă atributul sau grupul de atribute notat `CH`, care pentru T1 este cheie primară, iar pentru T2 – cheie străină, dacă în T2 se interzice apariţia de valori nenule ale `CH` care nu există în nici un tuplu din T1, se spune că între T2 şi T1 s-a instituit o restricţie referenţială.

Instituirea restricţiei referenţiale între tabelele TIP_BURSE (părinte) şi STUDENŢI (copil) permite cunoaşterea, pentru fiecare student, a denumirii bursei şi a cuantumului



Tip_burse

Tip_bursa	Den_bursa	Cuantum_bursă
M	Merit	1500000
S1	Studiu 1	1200000
S2	Studiu 2	900000

Studenti

Matricol	Numepren	Cnp	Adresa	Anstudiu	Modul_spec	Serieo	Grupa	Tip_bursa
M1	Topirceanu R Sorina	2222222222222	Str. Restantei nr. 13 bis, Mangalia, jud. Constant	1	2	4	2	NULL.
M2	Cojocaru W Simona	2222222222223	Str. Promovarii nr. 10, Iasi, jud. Iasi	1	2	4	2	M
M3	Zet T Marian	1222222222222	Str. Durerii nr. 145 bis, Focsani, jud. Vrancea	1	2	4	2	S1
M4	Vasilescu H Ioana	2222222222222	Str. Independentei nr. 4, Mangalia, jud. Constanta	1	2	4	2	S1
M5	Strat I Costel	1222222222223	.NULL.	1	2	4	2	NULL.
M6	Ionescu Y Vasile	1222222222224	Bd. Pacii, Iasi, jud. Iasi	3	I	1	1	NULL.
M7	Calencea J Neculai	1222222222225	Str. Unirii, Vaslui, jud. Vaslui	3	I	1	1	S2
M8	Manolache R Iolanda	2222222222226	.NULL.	3	C	1	2	S1
M9	Vasceuleanu G Ion	1222222222228	Bd. Ion Voda cel Cumplt, Suceava, jud. Suceava	1	2	4	2	S2
M10	Vlase G Georgel	1222222222229	Str. Macroeconomiei nr. 123, Iasi, jud. Iasi	1	1	1	1	S2
M11	Topliceanu T Ionela	2222222222210	Bd. Garii nr. 17, Roman, jud. Neamt	1	1	1	1	NULL.

Figura 2.2. Mecanismul de legare a tabelelor

lunar. Dacă în STUDENŢI ar exista vreo linie în care valoarea atributului `Tip_Burse` ar fi, spre exemplu S3, este clar că acea linie ar fi orfană (nu ar avea linie corespondentă în tabela-părinte).

Observaţii

- Pentru mulţi utilizatori şi profesionişti ai bazelor de date, denumirea de „relaţional" desemnează faptul că o bază de date este alcătuită din tabele puse în legătură prin intermediul cheilor străine. Aceasta este, de fapt, a doua accepţiune a termenului de BDR, prima, cea „clasică", având în vedere percepţia fiecărei linii dintr-o tabelă ca o relaţie între clase de valori.
- Majoritatea SGBD-urilor prezintă mecanisme de declarare şi gestionare automată a restricţiilor referenţiale, prin actualizări în cascadă şi interzicerea valorilor care ar încălca aceste restricţii.
- Respectarea restricţiilor referenţiale este una dintre cele mai complicate sarcini pentru dezvoltatorii de aplicaţii ce utilizează baze de date. Din acest punct de vedere, tentaţia este a „sparge" baza de date în cât mai puţine tabele cu putinţă, altfel spus, de a avea relaţii cât mai „corpolente". Gradul de fragmentare a bazei ţine de normalizarea bazei de date, care, ca parte a procesului de proiectare a BD, se bazează pe dependenţele funcţionale, multivaloare şi de joncţiune între atribute.

Restricţii-utilizator

Restricţiile-utilizator mai sunt denumite şi restricţii de comportament sau restricţii ale organizaţiei. De obicei, aceste restricţii iau forma unor reguli de validare la nivel de atribut, la nivel de linie sau de tabelă sau a unor reguli implementate prin declanşatoare

(*triggers*). Spre exemplu, se poate institui o regulă care interzice emiterea unei noi facturi (o nouă vânzare) dacă datoriile firmei client sunt mai mari de 2.000.000.000 lei, iar directorul acesteia nu este membru în partidul/partidele de guvernământ.

2.2. Nevoia de normalizare

Unul dintre cele mai bune argumente în favoarea normalizării ţine de punerea în evidenţă a ceea ce s-ar întâmpla în absenţa sa. Să luăm un prim exemplu, cel din figura 2.3, care este dedicat stocării datelor privitoare la studenţii Facultăţii de Economie şi Administrarea Afacerilor (sau oricare alta), mai ales în ceea ce priveşte situaţia şcolară a acestora. Fiecare student are un identificator unic - `Matricol` -, este înscris la o specializare într-un an de studii şi susţine examenele la disciplinele din planul de învăţământ. Orice disciplină are alocat un număr de credite prin planul de învăţământ al specializării. Fiecare student poate susţine un examen de mai multe ori, până îl promovează (sau este exmatriculat). Astfel, Zăineanu Ion a picat examenul la Baze de date I în prima sesiune (pe 29 ianuarie 2004), dar l-a luat în restanţe (pe 12 februarie).

STUDENŢI_EXAMENE

Matricol	NumePrenume	An	Specializare	CodDisc	DenumireDisc	NrCredite	DataExamen	Nota
EL13455	Popovici I Vasile	3	Informatică economică	AI3501	Baze de date I	6	29/01/2004	8
EL13456	Zăineanu W Ion	3	Informatică economică	AI3501	Baze de date I	6	29/01/2004	4
EL13457	Abălaşei R Zicu	3	Informatică economică	AI3501	Baze de date I	6	29/01/2004	9
EL13455	Popovici I Vasile	3	Informatică economică	AI3502	Programare vizuală şi RAD	7	01/02/2004	10
EL13456	Zăineanu W Ion	3	Informatică economică	AI3502	Programare vizuală şi RAD	7	01/02/2004	8
EL13457	Abălaşei R Zicu	3	Informatică economică	AI3502	Programare vizuală şi RAD	7	01/02/2004	4
EL13456	Zăineanu W Ion	3	Informatică economică	AI3501	Baze de date I	6	12/02/2004	8
EL13457	Abălaşei R Zicu	3	Informatică economică	AI3502	Programare vizuală şi RAD	7	15/02/2004	9
EL13458	Şpagă M Michael	3	Informatică economică	AI3503	Analiza sistemelor informaţionale	6	04/02/2004	7

Figura 2.3. Relaţie supusă normalizării

Deoarece fiecare linie se referă la un examen susţinut la o anumită dată de un student, cheia primară a relaţiei este combinaţia (`Matricol`, `CodDisc`, `DataExamen`).

2.2.1. *Redundanţe*

Este lesne de observat că relaţia de mai sus conţine redundanţe. Astfel, dacă un student susţine, pe parcursul primelor *n* semestre de studii, 25 de examene, atunci matricolul, numele, anul şi specializarea sunt prezente în toate cele 25 de linii. Dacă pentru o disciplină s-au consemnat în baza de date 1500 de examinări, atunci nu numai codul, ci şi denumirea şi numărul de credite alocat disciplinei apar de acelaşi număr de ori.

Cu cât baza de date este mai mare, cu atât risipa de spaţiu este mai importantă. Chiar dacă spaţiul de stocare nu mai este o problemă din punctul de vedere al costului, obezitatea unei tabele precum cea de mai sus poate atrage serioase probleme de viteză în exploatare (timpi de aşteptare la preluarea noilor note etc.).

2.2.2. *Anomalii la inserare*

Să luăm în discuţie studenţii din anul I. După examenul de admitere, ce are loc în lunile iulie şi septembrie, aceştia sunt înmatriculaţi. Dacă însă baza de date are schema relaţiei de mai sus, preluarea este imposibilă până în momentul primului examen al studentului respectiv. Aceasta deoarece în cheia primară sunt incluse şi atributele `CodDisc` şi `DataEx`, iar modelul relaţional interzice valori nule pentru atributele-cheie (restricţia de entitate). Or, la data înmatriculării, studenţii din anul I nu au nici un examen susţinut (asta ar mai trebui!), iar prima sesiune e abia în ianuarie, anul următor, aşa că şi `CodDisc`, şi `DataEx` sunt în acel moment NULL.

2.2.3. *Anomalii la modificare*

Presupunem că într-o furtunoasă şedinţă de catedră se decide ca disciplina *Programare vizuală şi RAD* să fie redenumită *Programare I*, titulatură sub care va apărea şi în foile matricole ale studenţilor din actualul an al III-lea, care vor absolvi, mai devreme sau mai târziu, specializarea Informatică economică. În baza de date există câteva sute de înregistrări referitoare la studenţii examinaţi la această disciplină, şi toate trebuie modificate. Dacă, dintr-un motiv sau altul, modificarea se face numai pe o parte dintre liniile cu pricina, putem afirma că datele îşi pierd consistenţa. După O'Neil & O'Neil, avem de-a face cu o anomalie de modificare într-o relaţie atunci când modificarea valorii unui atribut atrage obligativitatea actualizării aceleiaşi valori pe mai multe linii[4].

4. [O'Neil & O'Neil 01], p. 357.

2.2.4. *Anomalii la ştergere*

Anomaliile la ştergere se manifestă atunci când prin eliminarea unei linii dintr-o tabelă se pierd involuntar nu numai informaţiile despre entitatea reflectată de linia respectivă, ci şi alte informaţii. Astfel, dacă ştergem ultima linie din relaţia de mai sus, cea care conţine nota examenului la *Analiza sistemelor informaţionale* susţinut de studentul Şpagă Michael pe 4 februarie 2004, se pierd nu numai datele despre examenul respectiv, ci şi toate informaţiile despre studentul Şpagă M., deoarece aceasta era singura linie în care apărea studentul cu pricina.

2.3. Normalizarea – câteva tuşe

Văzută ca rigoare (matematică sau nu), artă (în sensul de „curat... interpretabilă", dar bazată şi pe intuiţie, creatoare), „meserie" (deprinsă prin imitaţie şi desăvârşită prin experienţă), ceea ce se poate spune cu siguranţă despre normalizare este, vorba lui C.J. Date, că nu reprezintă un panaceu.

Proiectarea bazelor de date relaţionale (BDR) presupune definirea atributelor, gruparea lor în tabele, stabilirea legăturilor dintre ele, a mecanismelor de integritate şi securitate, fiind un proces dificil în care cunoştinţele teoretice, experienţa, inteligenţa, intuiţia şi creativitatea proiectanţilor se îmbină într-un mod mai mult sau mai puţin armonios.

Structura finală a BD trebuie să asigure un echilibru între, pe de o parte, obţinerea unui volum maxim de informaţii într-un interval de timp cât mai scurt şi, pe de altă parte, eliminarea anomaliilor de stocare şi actualizare, toate în condiţiile unui grad apreciabil de securitate şi integritate.

Elaborarea schemei unei BD poate începe (şi) cu gruparea tuturor atributelor bazei de date într-o singură tabelă (relaţie universală). Dincolo de avantajul compactităţii, lucrul cu o singură (şi gospodărească) relaţie atotcuprinzătoare ridică serioase probleme privind redundanţa datelor şi generează anomalii importante la adăugarea, modificarea sau ştergerea unor linii (tupluri), după cum am văzut în frugalul exemplu din paragraful 2.1.

Ca o consecinţă directă, este necesară spargerea bazei în mai multe tabele care să fie legate prin restricţii de integritate referenţială. Este tocmai obiectivul central al normalizării. Spargerea relaţiei nu trebuie făcută oricum, deoarece apare riscul pierderilor de informaţii. De asemenea, un număr prea mare de tabele necesită un efort sporit pentru controlul bazei şi respectarea restricţiilor. În plus, trebuie să ţinem seama de faptul că obţinerea multor informaţii dintr-o bază de date fragmentată este posibilă prin operaţiunea de joncţiune, mare consumatoare de resurse.

Într-o enumerare mai mult decât seacă (deci ştiinţifică!), principalele obiective ale normalizării sunt:

- minimizarea spaţiului necesar stocării datelor;
- minimizarea riscului apariţiei de date inconsistente în cadrul BD;

- minimizarea anomaliilor ce pot apărea la actualizare (inserarea datelor, dar mai ales modificări şi ştergeri);
- ameliorarea structurii bazei, reprezentarea diverselor conexiuni dintre atributele bazei;
- diminuarea nevoii de reorganizare periodică a modelului.

Obiect al numeroase studii, nu se poate afirma că există o unanimitate de idei şi instrumente privind normalizarea. Importanţa normalizării nu trebuie absolutizată, deoarece aceasta vine adesea în contradicţie cu parametri extrem de importanţi ai aplicaţiilor de lucru cu bazele de date: viteză de acces, securitate mărită, resurse hard/soft disponibile, reducerea traficului pe reţea etc. Se poate spune că o BD riguros normalizată nu este neapărat una optimă.

Teoria clasică a normalizării este construită în jurul a cinci forme care în literatura noastră sunt referite ca *normale* [Lungu ş.a. 95] sau *normalizate* [Oprea 99]. Deşi termenul original este *normal form* (şi nu *normalized form*), mai româneşte sună *formă normalizată*. Nepunând prea mult preţ pe academisme, voi folosi în cele ce urmează ambele sintagme, fără resentimente. Pentru a fi într-o anumită formă normală/normalizată, o bază de date trebuie să respecte un anumit set de restricţii.

Codd, părintele modelului relaţional, a definit iniţial trei forme normale, notate prin 1NF, 2NF şi 3NF. Întrucât, într-o primă formulare, definiţia 3FN ridica ceva probleme, Codd şi Boyce au elaborat o nouă variantă, cunoscută sub numele *Boyce-Codd Normal Form* (BCNF). Deşi este vorba, în principiu, despre o formulare mai riguroasă a aceleiaşi 3FN, BCNF este prezentată separat în majoritatea lucrărilor.

Formele 4 şi 5, legate de numele lui Ronald Fagin, sunt tratate mai cu reţinere, ca să nu spun pudoare, în literatura consacrată analizei şi proiectării bazelor de date. Ba chiar unele lucrări cu tentă mai pragmatică se opresc declarat la 3FN, pe care o consideră suficientă în elaborarea BDR.

2.4. Etapele normalizării

Fundamentul normalizării BDR îl constituie dependenţele dintre atribute. Primele trei forme normale pot fi determinate pe baza dependenţelor funcţionale elementare (totale) şi tranzitive. Forma a patra (4FN) se bazează pe dependenţele multivaloare, în timp ce a cincea formă normală (5FN) – pe dependenţele de joncţiune. Problema este că dependenţele multivaloare şi (mai ales) cele de joncţiune sunt dificil de identificat şi rar întâlnite în practică.

Normalizarea BDR poate fi imaginată ca un proces prin care, pornindu-se de la relaţia iniţială R (universală), se realizează descompunerea succesivă a acesteia în subrelaţii, după succesiunea din figura 2.4[5]. Relaţia R poate fi ulterior reconstruită din cele *n* relaţii obţinute în urma normalizării, prin operaţii de joncţiune aplicate asupra acestora.

5. Vezi şi [Oprea 99], p. 343.

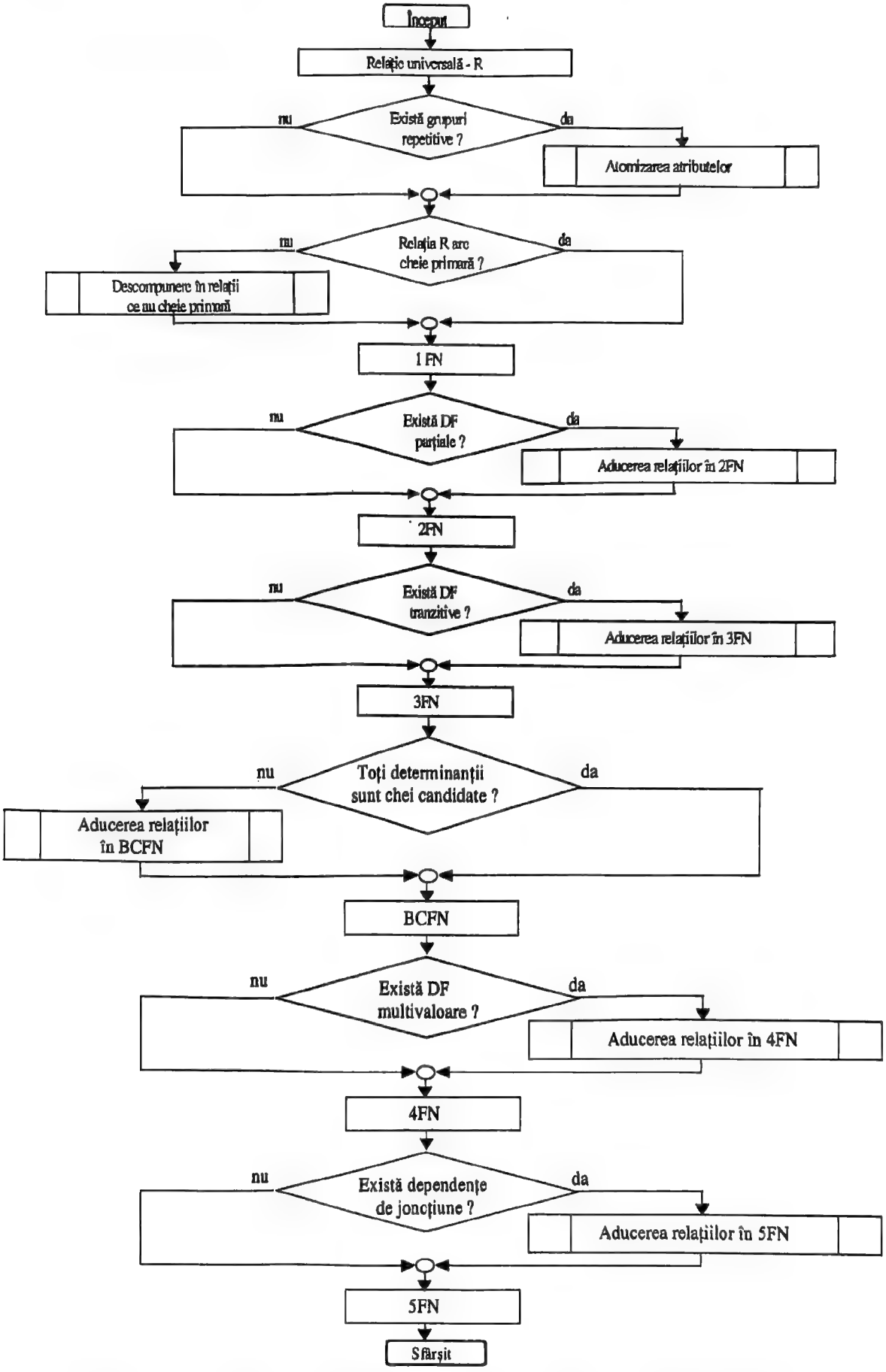


Figura 2.4. Etapele clasice ale normalizării bazelor de date relaționale

Nu putem încheia acest crochiu al normalizării fără a insista asupra faptului că, esenţialmente, ingredientul acesteia este de natură semantică. Normalizarea reflectă relaţii care se manifestă între atribute ale bazei. Este imposibil de normalizat o relaţie fără a cunoaşte fenomenul, procesul pe care îl reflectă.

2.5. Descompunere fără pierdere de informaţii

Pe baza elementelor discutate în paragraful 2.2, rezultă că prin normalizare se încearcă a se elimina redundanţe şi anomalii manifestate la actualizare. Eliminarea se realizează prin „sfărâmarea" relaţiei în două sau mai multe relaţii „legate" prin restricţii de integritate referenţială. Ulterior, prin operaţiuni de joncţiune, tabela iniţială poate fi recompusă, ori de câte ori va fi cazul.

Cu cât „desfacem" relaţia iniţială (universală) în mai multe tabele, cu atât redundanţa datelor este mai redusă (şi, implicit, anomaliile evocate). Există (şi aici) însă o limită, deoarece la descompunere trebuie avută în vedere păstrarea unor structuri care să permită, ulterior, obţinerea informaţiilor necesare prin operaţiuni de joncţiune (apare noţiunea de *joncţiune conservativă*).

Să presupunem că, în urma depistării redundanţei şi anomaliilor de inserare-modificare-ştergere, proiectantul bazei a ales soluţia descompunerii relaţiei STUDENŢI_EXAMENE (figura 2.3) în patru: STUDENŢI, DISCIPLINE, EXAMENE şi NOTE (figura 2.5).

STUDENŢI

Matricol	NumePrenume	An	Specializare
EL13455	Popovici I Vasile	3	Informatică economică
EL13456	Zăineanu W Ion	3	Informatică economică
EL13457	Abălaşei R Zicu	3	Informatică economică
EL13458	Spagă M Michael	3	Informatică economică

DISCIPLINE

CodDisc	DenumireDisc	NrCredite
AI3501	Baze de date I	6
AI3502	Programare vizuală şi RAD	7
AI3503	Analiza sistemelor informaţionale	6

EXAMENE

CodDisc	DataExamen
AI3501	29/01/2004
AI3502	01/02/2004
AI3501	12/02/2004
AI3502	15/02/2004
AI3503	04/02/2004

NOTE

Matricol	CodDisc	Nota
EL13455	AI3501	8
EL13456	AI3501	4
EL13457	AI3501	9
EL13455	AI3502	10
EL13456	AI3502	8
EL13457	AI3502	4
EL13456	AI3501	8
EL13457	AI3502	9
EL13458	AI3503	7

Figura 2.5. O descompunere a relaţiei universale STUDENŢI_EXAMENE

Redundanţele au fost reduse simţitor. Numele şi prenumele fiecărui student apar pe o singură linie; la fel şi denumirea disciplinei şi numărul de credite. Cu toate acestea, dacă încercăm să refacem relaţia iniţială prin interogarea SQL:

```
SELECT s.matricol, s.numeprenume, s.an,
s.specializare, d.coddisc, d.denumiredisc,
d.nrcredite, e.dataexamen, n.nota
FROM studenti s INNER JOIN note n ON s.matricol=n.matricol
   INNER JOIN discipline d ON d.coddisc=n.coddisc
   INNER JOIN examene e ON d.coddisc=e.coddisc
```

s-ar obţine tabela STUDENŢI_EXAMENE2 din figura 2.6.

STUDENŢI_EXAMENE2

Matricol	NumePrenume	An	Specializare	CodDisc	DenumireDisc	NrCredite	DataExamen	Nota
EL13455	Popovici I Vasile	3	Informatica economica	AI3501	Baze de date I	6	29.01.2004	8
EL13455	Popovici I Vasile	3	Informatica economica	AI3501	Baze de date I	6	12.02.2004	8
EL13456	Zaineanu W Ion	3	Informatica economica	AI3501	Baze de date I	6	29.01.2004	4
EL13456	Zaineanu W Ion	3	Informatica economica	AI3501	Baze de date I	6	12.02.2004	4
EL13457	Abalasei R Zicu	3	Informatica economica	AI3501	Baze de date I	6	29.01.2004	9
EL13457	Abalasei R Zicu	3	Informatica economica	AI3501	Baze de date I	6	12.02.2004	9
EL13455	Popovici I Vasile	3	Informatica economica	AI3502	Programare vizuala si RAD	7	15.02.2004	10
EL13455	Popovici I Vasile	3	Informatica economica	AI3502	Programare vizuala si RAD	7	01.02.2004	10
EL13456	Zaineanu W Ion	3	Informatica economica	AI3502	Programare vizuala si RAD	7	15.02.2004	8
EL13456	Zaineanu W Ion	3	Informatica economica	AI3502	Programare vizuala si RAD	7	01.02.2004	8
EL13457	Abalasei R Zicu	3	Informatica economica	AI3502	Programare vizuala si RAD	7	15.02.2004	4
EL13457	Abalasei R Zicu	3	Informatica economica	AI3502	Programare vizuala si RAD	7	01.02.2004	4
EL13456	Zaineanu W Ion	3	Informatica economica	AI3501	Baze de date I	6	12.02.2004	8
EL13456	Zaineanu W Ion	3	Informatica economica	AI3501	Baze de date I	6	29.01.2004	8
EL13457	Abalasei R Zicu	3	Informatica economica	AI3502	Programare vizuala si RAD	7	01.02.2004	9
EL13457	Abalasei R Zicu	3	Informatica economica	AI3502	Programare vizuala si RAD	7	15.02.2004	9
EL13458	Spaga M Michael	3	Informatica economica	AI3503	Analiza sistemelor informationale	6	04.02.2004	7
EL13455	Popovici I Vasile	3	Informatica economica	AI3501	Baze de date I	6	29.01.2004	8

Figura 2.6. Recompunerea relaţiei universale STUDENŢI_EXAMENE (obţinându-se o altă relaţie universală)

Se observă că rezultatul diferă sensibil de tabela inițială. Exemplul este un pic forțat, dar în practică deseori apar cazuri când erorile de proiectare a bazei sunt greu de identificat. Ceea ce s-a dorit a se demonstra este faptul că normalizarea nu înseamnă descompunerea arbitrară a tabelelor în altele mai mici.

2.6. Relația universală

Prima etapă a normalizării constă în identificarea informațiilor ce trebuie stocate și gestionate cu ajutorul bazei de date. Este momentul în care, la aplicațiile complexe, intră în scenă analiștii de sistem, cei care observă cum se derulează procesele, operațiunile ce constituie obiectul aplicației/bazei de date, discută cu toate categoriile de utilizatori, încearcă să identifice soluții anterioare de la aceeași organizație sau de la organizații similare pentru a putea anticipa problemele cele mai dificile etc. Pentru aplicațiile mari, munca analiștilor și proiectanților este extrem de laborioasă și consumatoare de timp și nervi. Este unul dintre motivele pentru care această categorie profesională are salarii mari și este destul de „curtată". Noi vom încerca să ne plasăm într-o zonă mai „aerisită", fără a cădea in derizoriu.

Noțiunea de relație universală este unul dintre destulele ingrediente ale modelului relațional care începe prin a părea jenant de banal și din care, în final, fiecare înțelege ce vrea, fără a ajunge la un punct comun, ci doar la un „nor" de puncte de vedere comune. Noi o să considerăm relația universală drept relația alcătuită din toate atributele identificate ca fiind relevante pentru aplicație și toate tuplurile posibile cu valorile acestor atribute[6]. Prin urmare, schema relației universale conține toate atributele bazei de date.

Baza de date VÂNZĂRI

Pe parcusul acestui capitol și al următoarelor ne propunem să elaborăm (și) schema unei prime baze de date prin care să poată fi gestionate informațiile relevante despre vânzările unei firme comerciale, de producție etc. Pentru o mai bună vizualizare, se poate recurge la o formă tabelară de prezentare a datelor, în care coloanele ar fi: numele atributului, așa cum este folosit în baza de date (ceva mai scurt), numele întreg al atributului, tipul atributului, lungimea, explicații suplimentare, restricții etc.

Tabelul 2.1. este ceva mai modest, având doar două coloane, dedicate numelui atributului și unor explicații sumare. Atributele care interesează în mod deosebit sunt cele legate de:

- *clienți*: codul, denumirea, adresa, codul poștal, localitatea, județul (inclusiv regiunea), codul fiscal, numărul de telefon;
- *produse*: codul, denumirea, unitatea de măsură, grupa (textile, încălțăminte, băuturi răcoritoare, băuturi spirtoase etc.), procentul de TVA aplicat;

6. Din acest punct de vedere, ne plasăm mai aproape de accepțiunea lui Chris Date ([Date 04], p. 195).

- *facturi*: numărul, data întocmirii, clientul, observații (note despre clauze din factură), apoi produsele vândute în factura respectivă: linia pe care apare fiecare produs, cantitatea și prețul facturat;
- încasări: codul unic asociat fiecărei încasări, date despre documentul pe baza căruia se face încasarea (ordin de plată, ceo etc.), numărul facturii încasate, tranșa încasată din factura respectivă.

Tabelul 2.1. VÂNZĂRI – dicționarul datelor (simplificat)

Atribut	Descriere
Jud	Indicativul auto al județului (2 caractere)
Județ	Denumirea județului
Regiune	Regiunea țării din care face parte județul
CodPost	Codul poștal al unei adrese
Localitate	Denumirea orașului sau satului
Comuna	Denumirea comunei din care face parte satul curent
CodCl	Codul firmei client
DenCl	Denumirea firmei client
CodFiscal	Codul fiscal al firmei client
StradaCl	Strada pe care se află sediul clientului
NrStradaCl	Numărul la care se afla adresa clientului
BlocScApCl	Eventualele informații suplimentare despre adresa clientului: bloc, scară, etaj, apartament
TelefonCl	Telefonul firmei client
CodPr	Codul produsului/sortimentului comercializat
DenPr	Denumirea produsului
UM	Unitatea de măsură a produsului/sortimentului
Grupa	Grupa sortimentală (Bere, Ciocolată, Cafea etc.) în care se încadrează produsul curent
ProcTVA	Procentul de TVA aplicat în mod obișnuit produsului
NrFact	Numărul facturii emise
DataFact	Data întocmirii facturii
Obs	Observații privitoare la factură
Linie	Linia curentă (corespunzătoare unui produs) dintr-o factură
Cantitate	Cantitatea vândută din produs în factură (pe linia curentă)
PreţUnit	Prețul unitar de vânzare (fără TVA) al produsului în factură (pe linia curentă)
CodÎnc	Codul încasării
DataÎnc	Data încasării (în care banii intră în casierie sau în contul bancar)
CodDoc	Codul documentului de încasare
NrDoc	Numărul documentului de încasare
DataDoc	Data la care a fost întocmit documentul de încasare
Tranșă	Tranșa primită din factură prin încasarea curentă (documentul curent)

Una dintre cele mai importante observaţii ce se cuvine a fi făcută în acest moment ţine de faptul că o serie de informaţii foarte importante, precum:

- valoarea fără TVA a unui produs facturat (de pe o linie a fiecărei facturi);
- TVA calculată pe linia unei facturi (pentru un produs vândut);
- valoarea cu TVA a unei linii (produs) dintr-o factură;
- valoarea fără TVA a unei facturi;
- TVA colectată aferentă unei facturi;
- valoarea totală (cu TVA) a unei facturi;
- valoarea totală a încasărilor pentru o factură;
- valoarea totală a vânzărilor către fiecare client;
- valoarea totală a încasărilor de la un client;
- restul de plată pentru fiecare client (diferenţa vânzări-încasări);
- valoarea vânzărilor pentru fiecare localitate;
- valoarea vânzărilor pentru fiecare judeţ;
- valoarea vânzărilor pentru fiecare regiune (Moldova, Banat etc.),

şi multe altele au fost eliminate din dicţionarul de date, deoarece pot fi calculate pornind ce la cele prezente în dicţionar. Vom vedea ceva mai târziu că, atunci când este vorba de date accesate în mod frecvent, pentru calcularea cărora este necesar un consum de resurse important, se poate recurge la introducerea controlată a unor atribute redundante.

O factură conţine mai multe linii şi poate fi încasată în una, două sau mai multe tranşe. Deoarece există un interval între momentul întocmirii de către client a documentului de plată (`DataDoc`) şi momentul intrării efective a banilor în contul (sau casieria) firmei (`DataÎnc`), se folosesc ambele atribute.

Baza de date FILMOGRAFIE

Pentru un centru de închirieri video se pune problema constituirii unei baze de date cu informaţii despre filmele aflate pe casetele sau DVD-urile oferite (legal) spre închiriere. Fiecare film are un cod unic (identificator), iar atributele principale sunt cele din tabelul 2.2.

Tabelul 2.2. FILMOGRAFIE – dicţionarul datelor (simplificat)

Atribut	Descriere
IdFilm	Codul unic al filmului
TitluOriginal	Titlul în engleză, franceză etc., aşa cum apare la lansarea filmului
TitluRO	Traducerea românească a titlului original
AnLans	Anul lansării
Producători	Producătorul/producătorii filmului
Regizori	Regizorul/regizorii filmului
Distribuţie	Actorii şi rolurile interpretate în film
Genuri	Genul/genurile în care se încadrează filmul (horror, comedie etc.)
Premii	Numele premiului – tipul (Oscar, Leul de argint, Ursul de aur etc.), categoria (cel mai bun film, cel mai bun actor în rol principal), anul premiului şi, eventual, numele actorului

Un film poate fi produs şi regizat de una, două sau mai multe persoane. Pentru fiecare actor distribuit în film interesează şi personajul sau personajele (dacă actorul e în dublu rol) interpretate. O peliculă se poate încadra la mai multe categorii (dramă, comedie) simultan. Dacă un premiu se referă la regie, scenariu etc., interesează numele, tipul şi anul decernării, iar dacă se referă la o performanţă actoricească, la informaţiile anterioare se adaugă şi numele actorului laureat.

Practic, două tupluri din această relaţie s-ar prezenta ca în figura 2.7.

IdFilm	TitluOriginal	TitluRO	AnLans	Producători	Regizori
11899	As Good As It Gets	Mai bine nu se poate	1997	James Brooks Bridget Johnson Kristi Zea	James Brooks
12345	Bicentennial Man	Omul bicentenar	1999	Michael Barnathan Chris Columbus Gail Katz	Chris Columbus

	Distribuţie	Genuri	Premii
	Jack Nicholson (Levin Udall) Helen Hunt (Carol Connelly) Greg Kinnear (Simon Bishop) Cuba Gooding Jr. (Frank Sachs)	comedie dramă romantic	Oscar – cel mai bun actor în rol principal (Jack Nicholson) – 1998 Oscar – cea mai bună actriţă în rol principal (Helen Hunt) – 1998 Globul de aur – cea mai bună imagine – 1998 Globul de aur – cel mai bun actor într-o comedie/musical (Jack Nicholson) – 1998 Globul de aur – cea mai bună actriţă într-o comedie/musical (Helen Hunt) – 1998
	Robin Williams (Andrew Martin) Embeth Davidtz (Little Miss Amanda Martia/Portia Charney) Sam Neil (Richard Martin) Oliver Platt (Rupert Burns) Kiersten Warren (Galatea)	SF dramă romantic	

Figura 2.7. O descompunere a relaţiei universale FILMOGRAFIE

Aceste două relaţii universale, ca şi altele, vor fi supuse, în capitolele următoare, normalizării şi tuturor discuţiilor şi problemelor legate de acest demers.

Încheiem acest capitol prin a mărturisi că, în ciuda relativei accesibilităţi a noţiunilor expuse vizavi de relaţia universală, literatura de specialitate a consemnat câteva dispute pe acest subiect. Spre exemplu, în 1981, William Kent afirma că includerea tuturor coloanelor într-o singură relaţie universală forţează folosirea valorilor nule[7]. Aşa cum vom discuta spre finalul capitolului următor, chiar dacă ne-am număra printre liber-null-işti (cei care sunt de acord cu folosirea valorilor NULL fără prea multă strângere de inimă, printre care se numără însuşi E.F. Codd),

7. [Kent 81].

de cele mai multe ori relaţia universală prezintă atribute componente ale cheii sale primare ce riscă să ia valori nule, ceea ce modelul relaţional nu acceptă, indiferent de tabere.

În articolul său din 1981, Kent este chiar tranşant, spunând că relaţia universală este un model nesatisfăcător pentru teoria şi practica relaţională[8]. Afirmaţia sa a stârnit reacţia a o serie de autori, printre care Jeffrey Ullman[9], pentru care, dincolo de inadvertenţe, conceptul de relaţie universală este unul polimorfic, fiind suficient de amintit două dintre abordări : pe de o parte, ca un soi de creuzet, rezervor al tuturor atributelor şi dependenţelor dintre atribute şi, pe de altă parte, ca o perspectivă generală a utilizatorilor asupra bazei de date.

Fără a-l supăra prea mult pe Ullman, vom considera relaţia universală drept o construcţie ipotetică de pornire în demersul normalizării, fără a fi anxioşi privitor la unicitatea sau neunicitatea sa şi alte proprietăţi care nu au prea multă tangenţă cu proiectarea efectivă a unei baze de date.

8. [Kent 81].
9. Vezi, spre exemplu, [Ullman 82] şi [Ullman 83].

CAPITOLUL 3

Prima formă normalizată

Prima formă normalizată (1NF) este, în general, tratată cu superficialitate de majoritatea practicienilor în materie de proiectare a bazelor de date. Aceasta deoarece principala cerinţă - atomicitatea valorilor - este un deziderat uşor de îndeplinit, cel puţin la prima vedere. Cele mai sensibile chestiuni în aducerea relaţiilor în 1NF sunt legate de atomicitate, eliminarea grupurilor repetitive, precum şi de stabilirea cheii primare, care devine o problemă delicată atunci când se porneşte de la o relaţie cu zeci sau sute de atribute.

Pe de altă parte, noţiunea de atomicitate este una destul de alunecoasă. Deşi în primele lucrări Codd nu formulează o definiţie riguroasă a atomicităţii, timp de aproape trei decenii, majoritatea autorilor din „zona" relaţionalului au considerat ca pe o primă „poruncă" a unei relaţii (tabele) necesitatea ca valoarea oricărui atribut pe orice linie (tuplu) să fie una scalară (atomică, nedecompozabilă). De câtva timp însă, Chris Date şi alţi relaţionişti încearcă că depăşească limitele atomicităţii, înlocuind domeniile cu *tipuri* care pot fi atât scalare, cât şi compozite (definite de utilizator).

3.1. Definiţii ale primei forme normalizate

Teoria normalizării nu face parte, propriu-zis, din modelul relaţional. Dintre toate formele normalizate, doar prima are caracter de obligativitate. O bază de date este normalizată dacă toate relaţiile (tabelele) care o alcătuiesc se află *măcar* în 1NF. Celelalte forme normale sunt dezirabile pentru diminuarea redundanţelor, spaţiului ocupat pe disc sau anomaliilor manifestate la actualizare, însă nu sunt obligatorii.

După Codd, *o relaţie este în prima formă normală dacă* [...] *nici unul dintre domeniile sale nu conţine elemente care sunt, la rândul lor, seturi (ansambluri)*[1]. Se cuvine de adăugat remarca lui Date care crede că ar fi fost mai nimerit ca, în locul termenului *domenii*, Codd să-l fi folosit pe cel de *atribute*. Actualmente, mulţi „relaţionişti" preferă termenul *tip* celui de *domeniu*[2].

1. [Codd 72].
2. [Date 04].

Unele lucrări[3] definesc o relaţie aflată în 1NF ca *acea relaţie în care fiecare atribut prezintă numai valori atomice*, adică *toate atributele sunt nedecompozabile*; după R. Riordan, o relaţie este în 1NF dacă *domeniile pe care sunt definite atributele relaţiei sunt scalare*[4]. În ediţia a treia a cărţii dedicate sistemelor de baze de date, Connoly şi Begg sunt mai tranşanţi: *o relaţie în 1NF este o relaţie în care intersecţia oricărei linii cu oricare coloană conţine o valoare şi numai una*[5]. Astfel, 1NF respinge ideea de set de valori, tuplu de valori sau combinaţia acestora ca valoare a unui singur atribut pe un oricare tuplu (linie) a relaţiei.

Alţi autori formulează explicit pentru 1NF şi restricţia: *toate atributele ce compun relaţia să fie în dependenţă funcţională faţă de cheia relaţiei*, ceea ce este oarecum o tautologie, întrucât una dintre „poruncile" valabile pentru orice tabelă stipulează negru pe alb că într-o relaţie nu pot exista două tupluri identice sau, altfel spus, într-o relaţie trebuie să existe măcar un atribut sau combinaţie de atribute ale căror valori să deosebească orice tuplu de toate celelalte sau orice relaţie posedă o cheie primară.

Formularea care a generat cele mai multe confuzii este legată de noţiunea de *grupuri repetitive*[6]: o relaţie în 1NF nu trebuie să conţină grupuri repetitive.

De obicei, noţiunea de grup repetitiv este raportată direct la cea de atomicitate. Diferenţierea atomicitate-grup repetitiv apare în [Lungu ş.a. 95], pentru care un atribut atomic este unul simplu (necompus): „O relaţie este în 1NF dacă domeniile pe care sunt definite atributele relaţiei sunt constituite numai din valori atomice (elementare). În plus, un tuplu nu trebuie să conţină atribute sau grupuri de atribute repetitive"[7]. Din păcate, autorii se opresc aici cu definiţia, păstrând o remarcabilă discreţie vizavi de ceea ce consideră grup repetitiv. La fel stau lucrurile şi la dna Ileana Popescu[8].

Un plus de claritate aduce definiţia lui Toby Teorey, pentru care o relaţie este în 1NF dacă şi numai dacă toate coloanele conţin numai valori atomice; aceasta înseamnă că *nu există grupuri (coloane) repetitive în cadrul vreunei linii* a tabelei. Imediat autoarea precizează că *un grup repetitiv apare într-o tabelă atunci când unui atribut multi-valoare îi sunt permise două sau mai multe valori reprezentate în cadrul unei aceleiaşi linii*[9]. Sau, după cum spunea Chris Date până în ediţia a şasea a celei mai cunoscute cărţi a sa, un grup repetitiv reprezintă o coloană care conţine câteva valori în fiecare linie sau, altfel spus, un număr diferit de valori pe linii diferite[10].

3. [Date 86], [Miranda & Busta 90], [Connoly ş.a. 96], [Dollinger 98].
4. [Riordan 99].
5. [Connoly & Begg 02], p. 388.
6. [Date 86], [Pratt & Adamski 91], [Oprea 99], [Lungu ş.a. 95].
7. [Lungu ş.a. 95], p. 169.
8. [Popescu 01], p. 78.
9. [Teorey 99], p. 99.
10. Preluare din ediţia a opta a cărţii ([Date 04], p. 153), unde Date se autocitează pentru a demonstra că înţelesese greşit noţiunea de domeniu (tip) şi, în consecinţă, greşise atunci când a formulat definiţia 1NF.

În concluzie, aducerea unei relaţii în 1NF presupune discutarea următoarelor elemente:

- raportul atribut simplu/atribut compus;
- grupuri repetitive de atribute (pe orizontală);
- grupuri repetitive de valori în cadrul fiecărei celule a tabelei.

3.2. Atribute simple şi atribute compuse

Primele vizate de „stigmatul" neatomicităţii sunt atributele compuse. Cu ani buni în urmă, în cursurile de profil, exemplul clasic al unui atribut compus era `DataCalendaristică`, alcătuit, cum altfel, din câmpurile: `Zi`, `Lună`, `An`. Era vremea înfloritoare a COBOL-ului şi FORTRAN-ului, când nu existau mecanisme de declarare şi gestionare a variabilelor şi atributelor de tip DATE; verificarea corectitudinii datei calendaristice cădea în sarcina programatorului, care trebuia să scrie o rutină specială.

Astăzi, nu există SGBD sau mediu de programare care să nu prezinte tipul de dată DATE, prin care se gestionează simultan toate cele trei elemente; prin funcţii de conversie se pot obţine: ziua (DAY), luna (MONTH), anul (YEAR), ba chiar calcula numărul de zile, luni, ani dintre două date calendaristice şi alte minunăţii. Ca să nu mai vorbim de tipul DATETIME, şi acesta omniprezent în produsele software ale zilelor noastre.

Un alt exemplu celebru privind atribute compuse (non-atomice) este `Adresa`. Fiind alcătuită din `Stradă`, `Număr`, `Bloc`, `Scară`, `Etaj`, `Apartament`, plus `CodPoştal` (după noul sistem de codificare), discuţia despre atomicitatea adresei pare de prisos, iar descompunerea sa imperativă. Din nou însă trebuie să ne raportăm la obiectivele bazei. Fără îndoială că, pentru BD a unei filiale CONEL sau ROMTELECOM, sau pentru Poliţie, preluarea separată a fiecărui element constituent al adresei este vitală. Interesează, spre exemplu, ce abonaţi sunt pe strada Tranziţiei sau câte cereri sunt depuse de cetăţenii de pe străzile Lascăr Catargi, Titu Maiorescu şi Marina Scupra. Sau, un cioranian ce lucrează în cadrul Poliţiei ar putea fi interesat să afle dacă rata sinuciderilor la persoanele ce locuiesc la etajul trei este mai mare decât la persoanele de la parter.

Cu totul altfel stau lucrurile pentru un importator direct, pentru un mare angrosist sau pentru o firmă de producţie. Partenerii de afaceri ai acestora sunt persoane juridice, iar adresa interesează numai la nivel general. Este drept că imediat după Revoluţie deveniseră mari furnizori naţionali de cupru şi aluminiu şi persoane fizice (vezi cazul bulibaşei din Ciurea, dar asta-i altă poveste!).

Alte exemple de atribute ce pot fi considerate, în funcţie de circumstanţe, simple sau compuse: `DataOperaţiuniiBancare (Data+Ora)`, `BuletinIdentitate (Seria+Număr)`, `NrÎnmatriculareAuto` (privit global, sau pe cele trei componente: judeţ, număr şi combinaţia de trei litere).

În relaţia universală FILMOGRAFIE, al cărei dicţionar de date simplificat este cel din tabelul 2.2, cel puţin două atribute sunt compuse şi trebuie sparte, `Distribuţie`

şi `Premii`. Astfel, în `Distribuţie` pot fi delimitate măcar două informaţii, `Actor` şi `Rol`. În cazul premiilor, ar trebui detaliată discuţia prin introducerea atributelor :

- denumire premiu (*Oscar*, *Globul de aur* etc.) ;
- locul atribuirii (Hollywood, Cannes, Berlin, Veneţia, Bucureşti etc.) ;
- categoria (regie, scenariu, actor în rol principal etc.) ;
- anul decernării ;
- numele actorului, în cazul categoriei interpretare.

Astfel, noua formă a dicţionarului de date este cea din tabelul 3.1.

Tabelul 3.1. FILMOGRAFIE – dicţionarul datelor (simplificat), versiunea 2

Atribut	Descriere
IdFilm	Codul unic al filmului
TitluOriginal	Titlul în engleză, franceză etc., aşa cum apare la lansarea filmului
TitluRO	Traducerea românească a titlului original
AnLans	Anul lansării
Producători	Producătorul/producătorii filmului
Regizori	Regizorul/regizorii filmului
Rol	Rolul din film
Actor	Actorul care interpretează rolul curent
Genuri	Genul/genurile în care se încadrează filmul (horror, comedie etc.)
DenPremiu	Numele premiului – tipul (Oscar, Leul de argint, Ursul de aur etc.)
LocDecernare	Locul în care se organizează festivitatea sau festivalul
Categorie	Categoria premiului (pentru ce anume se acordă premiul)
AnPremiu	Anul decernării

Concluzionând în termeni rabinici, ambele variante, descompunerea sau nedescompunerea atributelor compuse, sunt valabile de la caz la caz, alegerea soluţiei fiind în concordanţă cu specificul bazei de date (şi inspiraţia proiectantului). În plus, nu trebuie uitat că şi pentru atributele compuse pot fi extrase destule informaţii utilizând în SQL operatorul `LIKE`.

Atomicitatea atributelor într-o bază de date relaţională a fost privită ca o limită serioasă a modelului, datorită imposibilităţii adaptării BDR la specificul unor domenii precum multimedia, CAD (*Computer-Aided Design*) etc.

Mulţi autori, dintre care merită amintiţi cu deosebire Chris J. Date şi Hugh Darwen, se opun ideii de atomicitate formulate de Codd (vezi şi paragraful 3.5).

3.3. Despre grupuri repetitive şi urmările lor

O relaţie (tabelă) în 1NF nu trebuie să conţină atribute care se repetă ca grupuri. Într-o altă formulare, toate liniile unei tabele trebuie să conţină acelaşi număr de atribute. Fiecare celulă a tabelei (intersecţia unei coloane cu o linie), altfel spus, valoarea unui atribut pe o linie (înregistrare), trebuie să fie atomică.

După [Connoly ş.a. 96], un grup repetitiv este un atribut sau grup de atribute dintr-o tabelă care apare cu valori multiple pentru o singură apariţie a cheii primare a tabelei nenormalizate.

Să luăm exemplul tabelei BIBLIOTECĂ din figura 3.1. Tabela gestionează informaţii despre cărţile existente în depozitul bibliotecii Facultăţii de Economie şi Administrarea Afacerilor (FEAA). De remarcat că în bliblioteca există două exemplare ale cărţii cu ISBN-ul 973-683-709-2 şi trei exemplare de cea dedicată Visual FoxPro. Prima carte a fost scrisă de patru autori şi ei îi sunt asociate opt cuvinte-cheie, iar a doua are un singur autor. Relaţia nenormalizată conţine trei grupuri repetitive: `Cote`, `Autori` şi `CuvinteCheie`.

BIBLIOTECĂ

ISBN	Titlu	Cote	Autori	Editura	LocSediuEd	AnApariţie	CuvinteCheie
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13421, III-13422, III-13423	Marin Fotache, Ioan Brava, Cătălin Strâmbei, Liviu Creţu	Polirom	Iaşi	2002	baze de date, SQL, proceduri stocate, FoxPro, formulare, orientare pe obiect, client-server, Web
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	III-10678, III-10679	Marin Fotache	Polirom	Iaşi	2001	baze de date, algebră relaţională, SQL

Figura 3.1. Relaţia universală nenormalizată BIBLIOTECĂ

Paragrafele următoarele sunt dedicate prezentării unor soluţii pentru eliminarea grupurilor repetitive. Trebuie precizat însă că nu întotdeauna grupurile repetitive reprezintă o problemă majoră. Spre exemplu, într-o relaţie gen PERSOANE_CONTACT, atributul `Telefoane` poate avea valori neatomice, deoarece pot exista mai multe numere atât la birou, cât şi de mobile (mulţi manageri au câte un aparat pe fiecare reţea – Orange, Connex, Zapp).

3.3.1. *Grupuri repetitive pe orizontală*

Una dintre cele mai nerecomandate soluţii pentru eliminarea grupurilor repetitive ţine de extinderea pe orizontală a tabelelor, prin duplicarea forţată a unor atribute. Revenim la figura anterioară. Pentru a înlătura grupurile repetitive de pe primele două linii, putem fi tentaţi să modificăm structura tabelei, ca în figura 3.2.

Deşi tabela se află în prima formă normală, structura acesteia este hilară. Ar trebui să „rezervăm" câteva zeci de atribute pentru cotele cărţilor, altă serie de atribute pentru autori (apropo, care este numărul maxim de autori ce au participat la scrierea unei cărţi?), ca să nu mai vorbim de cuvintele-cheie.

Atenţie, însă! Deşi hilară, schema de mai sus nu violează conceptul de atomicitate, oricare ar fi formularea acestuia. Unii autori respectabili, precum Toby Teorey[11],

11. [Teorey 99], pp. 99-100.

BIBLIOTECĂ_GRUPURI_REPETITIVE_PE_ORIZONTALĂ

ISBN	Titlu	Cota1	Cota2	Cota3
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13421	III-13422	III-13423
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	III-10678	III-10679	NULL

Autor1	Autor2	Autor3	Autor4	Editura	LocSediuEd
Marin Fotache	Ioan Brava	Cătălin Strâmbei	Liviu Creţu	Polirom	Iaşi
Marin Fotache	NULL	NULL	NULL	Polirom	Iaşi

AnApariţie	CuvântCheie1	CuvântCheie2	CuvântCheie3	CuvântCheie4
2002	baze de date	SQL	proceduri stocate	FoxPro
2001	baze de date	algebră relaţională	SQL	NULL

CuvântCheie5	CuvântCheie6	CuvântCheie7	CuvântCheie8
formulare	orientare pe obiect	client-server	Web
NULL	NULL	NULL	NULL

Figura 3.2. O soluţie (destul de penibilă) de eliminare a grupurilor repetitive

greşesc atunci când dau exemple de relaţii nenormalizate de tipul celor din figura 3.2 (este drept, la Teorey, atributele care se repetă în cadrul relaţiei au acelaşi nume, ceea ce modelul relaţional respinge din start).

Această variantă de normalizare funcţionează totuşi în cazuri punctuale. Spre exemplu, nu este exagerat ca într-o tabelă să avem atribute de genul celui din figura 3.3.

PERSOANE_1

CNP	Nume	Prenume	DataNaşterii	CNPMamă	CNPTată
2641121390802	Bucur	Cerasela	21-11-1964	2440611390167	1401205390102
...					

Figura 3.3. Un exemplu rezonabil de atribut pseudo-repetitiv

Cum orice persoană are cel mult doi părinţi (re)cunoscuţi, putem introduce grupul de atribute `CNPMamă, CNPTată`, deşi, din punct de vedere relaţional, este corectă şi varianta din figura 3.4.

PERSOANE_2

CNP	Nume	Prenume	DataNaşterii	CNPPărinte	Părinte
2641121390802	Bucur	Cerasela	21-11-1964	2440611390167	Mamă
2641121390802	Bucur	Cerasela	21-11-1964	1401205390102	Tată
...					

Figura 3.4. O altă primă formă normală a relaţiei PERSOANE_1

3.3.2. *Eliminarea grupurilor repetitive prin adăugarea de tupluri*

A doua variantă păstrează structura relaţiei (universale sau nu), în sensul că nu se modifică numărul de atribute, adăugându-se tupluri (linii) suplimentare astfel încât orice valoare a celor trei atribute de tipul „grupuri repetitive" să devină atomică. Apare însă problema : cum anume completăm tuplurile pentru ca să fie preluate toate informaţiile, iar, pe de altă parte, eventualele actualizări să nu antreneze pierderi sau alterări de informaţii ? În acest sens, am putea lua în discuţie trei soluţii, după cum urmează :

Număr minim de linii – o singură valoare atomică pentru fiecare atribut compozit

Avem trei atribute non-atomice (multivaloare). În figura 3.1, pentru fiecare carte, numărul de linii din tabelă va fi maximul dintre numărul cotelor, numărul autorilor şi numărul cuvintelor-cheie. Astfel, pentru *Visual FoxPro* sunt trei cote, patru autori

BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_1

ISBN	Titlu	Cotă	Autor
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13421	Marin Fotache
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13422	Ioan Brava
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13423	Cătălin Strîmbei
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	Liviu Creţu
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	III-10678	Marin Fotache
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	III-10679	NULL
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	NULL	NULL

Editura	LocSediuEd	AnAparitie	CuvântCheie
Polirom	Iaşi	2002	baze de date
Polirom	Iaşi	2002	SQL
Polirom	Iaşi	2002	proceduri stocate
Polirom	Iaşi	2002	FoxPro
Polirom	Iaşi	2002	formulare
Polirom	Iaşi	2002	orientare pe obiect
Polirom	Iaşi	2002	client-server
Polirom	Iaşi	2002	Web
Polirom	Iaşi	2001	baze de date
Polirom	Iaşi	2001	algebră relaţională
Polirom	Iaşi	2001	SQL

Figura 3.5. Completarea cu tupluri – soluţia 1

şi opt cuvinte-cheie ; aşadar, vom avea opt linii (primele opt din figura 3.5). Valorile cotelor în liniile 4-8 vor fi NULL ; la fel şi valorile atributului `Autor` pentru liniile 5-8.

Deoarece o cotă nu poate identifica decât o singură carte, cheia primară a noii relaţii este alcătuită din trei atribute (`Cotă`, `Autor`, `CuvântCheie`). Redundanţa datelor este evidentă. Dacă pentru cartea de Visual FoxPro am dori să mai introducem un nou cuvânt-cheie, ar trebui inserată o linie nouă în tabelă, în care valorile atributelor `Autor` şi `Cotă` să fie nule. Aceasta nu este însă singura problemă. Să presupunem că sintagma *proceduri stocate* este greşit asociată cărţii de Visual FoxPro. În mod normal, corectarea acestei greşeli presupune ştergerea celui de-al treilea tuplu :

```
DELETE FROM BIBLIOTECĂ_TUPLURI_NOI_SOLUTIA_1
WHERE ISBN = '973-683-889-7' AND
cuvântcheie = 'proceduri stocate')
```

Necazul e că ştergerea liniei se face cu două pierderi de informaţii. Prima pierdere ţine de calitatea de coautor al acestei cărţi a lui Cătălin Strîmbei, iar a doua este legată de exemplarul cărţii ce are cota III-13423. Aceasta deoarece tuplul şters era singurul care se referea la autorul şi la cota cu pricina.

Am putea încerca un artificiu, şi anume ca ştergerea unui cuvânt-cheie, a unui autor ori a unei cote să nu se facă prin ştergerea liniei respective, ci prin setarea pe NULL a valorii de pe linia incriminată :

```
UPDATE BIBLIOTECĂ_TUPLURI_NOI_SOLUTIA_1
SET cuvântcheie = NULL
WHERE ISBN = '973-683-889-7'
AND cuvântcheie = 'proceduri stocate')
```

Nici această idee nu e din cale-afară de inteligentă. Dacă am dori eliminarea sintagmei *Web* pentru aceeaşi carte :

```
UPDATE BIBLIOTECĂ_TUPLURI_NOI_SOLUTIA_1
SET cuvântcheie = NULL
WHERE ISBN = '973-683-889-7' AND cuvântcheie = 'Web')
```

atunci linia modificată devine cu totul de prisos în tabelă, toate cele trei atribute, `Cota`, `Autor` şi `CuvântCheie`, având valori NULL. Cu alte cuvinte, după modificare, tuplul nu spune nimic, nu are valoare informaţională. Practic, după fiecare NULL-izare pe o linie a unuia dintre cele trei atribute, trebuie verificat dacă pe linia respectivă celelalte două atribute au valori NULL, caz în care linia se poate şterge.

Mai există o problemă şi din punct de vedere al conţinutului informaţional. Să presupunem că ne interesează să aflăm care sunt autorii ce au scris despre SQL. Ţinând seama de stuctura relaţiei de mai sus, am fi tentaţi să folosim o selecţie după cuvântul-cheie SQL, urmată de o proiecţie a valorilor atributului `Autor` :

```
SELECT autor
FROM BIBLIOTECĂ_TUPLURI_NOI_SOLUTIA_1
WHERE cuvântcheie = 'SQL')
```

Rezultatul furnizat este alcătuit din două linii, una ce conţine valoarea *Ioan Brava*, iar cealaltă valoarea NULL. Lipsesc deci ceilalţi trei autori ai cărţii despre Visual FoxPro, dintre care unul (săracul!) a scris chiar o carte despre SQL. Interogarea ce furnizează răspunsul corect trebuie să folosească o joncţiune sau corelare după atributul `ISBN`. Prin urmare, este necesar de ştiut că structura informaţională pe care o reprezintă tabela de mai sus are o serie de neajunsuri.

Cel mai bine întemeiat motiv de renunţare la această variantă este prezenţa valorilor nule în componenţa atributelor chei primare, după cum vom discuta în paragraful 3.4.

Număr de linii egal cu suma valorilor elementare ale atributelor de tip grup repetitiv

A doua soluţie de completare are în vedere eliminarea problemei care apare la ştergerea unei linii în cazul soluţiei precedente, şi anume: pe oricare linie, o valoare nenulă a unuia dintre cele trei atribute atrage după sine nulitatea celorlalte două - vezi figura 3.6. Spre exemplu, prima carte are 3 cote, 4 autori şi 8 cuvinte-cheie; prin urmare, vor fi necesare 3 + 4 + 8 = 15 linii.

Această secundă soluţie, deşi presupune un număr mai mare de linii, elimină necazul pierderii involuntare de informaţii. Astfel, ştergerea unui cuvânt-cheie nu va afecta, cu siguranţă, nici un autor sau exemplar al cărţii respective.

Ca şi în cazul soluţiei anterioare, indezirabilitatea variantei este legată de prezenţa valorilor nule în componenţa atributelor chei primare.

Număr maxim de linii, egal cu produsul valorilor elementare ale atributelor de tip grup repetitiv

De data aceasta, tăiem valorile nule de la rădăcină, introducând în relaţie toate combinaţiile `Autor-Cotă-CuvântCheie`. Cum prima carte are 3 cote, 4 autori şi 8 cuvinte-cheie, în relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 vor exista 3 * 4 * 8 = 96 de tupluri referitoare la aceasta - vezi figura 3.7.

Cele opt linii din figură se referă doar la primele două cuvinte-cheie (*baze de date* şi *SQL*) pentru primul exemplar (cota III-13421) din prima carte. Aducerea relaţiei în 1NF eliminând grupurile repetitive prin adăugarea de noi tupluri de această manieră se dovedeşte mai mult decât costisitoare. Aceasta nu numai din punctul de vedere al spaţiului excesiv consumat din pricina redundanţei masive, dar şi dacă ne gândim la anomaliile de actualizare. Spre exemplu, pentru a nu pierde informaţii, atunci când adăugăm un cuvânt-cheie nou pentru cartea luată spre analiză, e nevoie să inserăm în tabelă nu mai puţin de 3 (cote) * 4 (autori) = 12 linii. Dezvoltând această idee optimistă, imaginaţi-vă cum stau lucrurile pentru o lucrare monumentală cu 10 autori şi 78 de cuvinte-cheie din care în bibliotecă există 40 de exemplare!

BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_2

ISBN	Titlu	Cotă	Autor
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13421	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13422	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	III-13423	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	Marin Fotache
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	Ioan Brava
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	Cătălin Stâmbei
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	Liviu Creţu
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	NULL	NULL
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	III-10678	NULL
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	III-10679	NULL
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	NULL	Marin Fotache
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	NULL	NULL
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	NULL	NULL
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	NULL	NULL

Editura	LocSediuEd	AnApariţie	CuvântCheie
Polirom	Iaşi	2002	NULL
Polirom	Iaşi	2002	NULL
Polirom	Iaşi	2002	NULL
Polirom	Iaşi	2002	NULL
Polirom	Iaşi	2002	NULL
Polirom	Iaşi	2002	NULL
Polirom	Iaşi	2002	NULL
Polirom	Iaşi	2002	baze de date
Polirom	Iaşi	2002	SQL
Polirom	Iaşi	2002	proceduri stocate
Polirom	Iaşi	2002	FoxPro
Polirom	Iaşi	2002	formulare
Polirom	Iaşi	2002	orientare pe obiect
Polirom	Iaşi	2002	client-server
Polirom	Iaşi	2002	Web
Polirom	Iaşi	2001	NULL
Polirom	Iaşi	2001	NULL
Polirom	Iaşi	2001	NULL
Polirom	Iaşi	2001	baze de date
Polirom	Iaşi	2001	algebră relaţională
Polirom	Iaşi	2001	SQL

Figura 3.6. Completarea cu tupluri – soluţia 2

BIBLIOTECĂ_TUPLURI_NOI_SOLUȚIA_3 (fragment)

ISBN	Titlu	Cotă	Autor
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Marin Fotache
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Ioan Brava
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Cătălin Stâmbei
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Liviu Crețu
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Marin Fotache
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Ioan Brava
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Cătălin Stâmbei
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	III-13421	Liviu Crețu
	.... (încă 88 de linii pentru cartea de Visual FoxPro, plus 6 pentru cartea de SQL)		

Editura	LocSediuEd	AnApariție	CuvântCheie
Polirom	Iaşi	2002	baze de date
Polirom	Iaşi	2002	baze de date
Polirom	Iaşi	2002	baze de date
Polirom	Iaşi	2002	baze de date
Polirom	Iaşi	2002	SQL
Polirom	Iaşi	2002	SQL
Polirom	Iaşi	2002	SQL
Polirom	Iaşi ...	2002	SQL

Figura 3.7. Completarea cu tupluri – soluția 3

3.3.3. *Eliminarea grupurilor repetitive prin construirea de noi relații*

După năduful provocat de redundanța masivă a ultimei soluții de normalizare, cea care urmează este ca o ușurare: „ruperea" din relația universală a grupurilor și constituirea de relații noi[12]. Spre exemplu, în [Lungu ș.a. 95] este formulată „rețeta" de aducere a unei relații în 1NF prin parcurgerea a patru pași:

- în relația universală, în locul atributelor compuse se trec componentele acestora, ca atribute simple;

12. Soluția este sugerată în [Lungu ș.a. 95], [Oprea 99], [Connoly ș.a. 96], [Elmasri & Navathe 00] etc.

- se constituie câte o relaţie pentru fiecare grup repetitiv;
- în schema fiecărei noi relaţii obţinute la pasul doi se introduce şi cheia primară a relaţiei R nenormalizate;
- cheia primară a fiecărei noi relaţii va fi compusă din atributele/atributele chei ale R plus unul sau mai multe atribute proprii.

Aplicând acest algoritm la relaţia BIBLIOTECĂ, obţinem patru tabele, COTE, AUTORI_CĂRŢI, CĂRŢI_CUVCHEIE şi CĂRŢI, cu structura din figura 3.8.

COTE

ISBN	Cotă
973-683-889-7	III-13421
973-683-889-7	III-13422
973-683-889-7	III-13423
973-683-709-2	III-10678
973-683-709-2	III-10679

CĂRŢI_CUVCHEIE

ISBN	CuvântCheie
973-683-889-7	baze de date
973-683-889-7	SQL
973-683-889-7	proceduri stocate
973-683-889-7	FoxPro
973-683-889-7	formulare
973-683-889-7	orientare pe obiect
973-683-889-7	client-server
973-683-889-7	Web
973-683-709-2	baze de date
973-683-709-2	algebră relaţională
973-683-709-2	SQL

AUTORI_CĂRŢI

ISBN	Autor
973-683-889-7	Marin Fotache
973-683-889-7	Ioan Brava
973-683-889-7	Cătălin Strâmbei
973-683-889-7	Liviu Creţu
973-683-709-2	Marin Fotache
973-683-709-2	Marin Fotache

CĂRŢI

ISBN	Titlu	Editura	LocSediuEd	AnAparitie
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Polirom	Iaşi	2002
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Polirom	Iaşi	2001

Figura 3.8. O a patra tentativă de aducere în 1NF a relaţiei BIBLIOTECĂ

Structura obţinută se detaşează prin cel mai mic volum de redundanţă de până acum, ceea ce antrenează şi economii de stocare, şi reducerea sensibilă a anomaliilor de actualizare. Personal însă, reproşez acestui stil de normalizare bazat pe eliminarea grupurilor repetitive caracterul prea „ochiometric". Observarea grupurilor repetitive depinde decisiv de ordinea în care sunt dispuse atributele în relaţie (teoria relaţională prevede clar că nici ordinea coloanelor, nici cea a liniilor nu influenţează conţinutul informaţional al unei tabele).

Să discutăm relaţia BIBLIOTECĂ_2, ce are aceeaşi structură ca BIBLIOTECĂ, dar ordinea atributelor este schimbată - vezi figura 3.9.

Tabela BIBLIOTECĂ_2 este mai aproape de realitate decât BIBLIOTECĂ. În depozit sunt trei exemplare ale cărţii dedicate Visual FoxPro, acestora corespunzându-le trei tupluri în noua relaţie. Cheia primară este `Cotă` (observaţi că `ISBN`-ul se repetă pe primele trei linii şi pe ultimele două).

BIBLIOTECĂ_2

Cotă	ISBN	Titlu	Autori	Editura	LocSediuEd	AnAparitie	CuvinteCheie
III-13421	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Marin Fotache, Ioan Brava, Cătălin Strâmbei, Liviu Creţu	Polirom	Iaşi	2002	baze de date, SQL, proceduri stocate, FoxPro, formulare, orientare pe obiect, client-server, Web
III-13422	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Marin Fotache, Ioan Brava, Cătălin Strâmbei, Liviu Creţu	Polirom	Iaşi	2002	baze de date, SQL, proceduri stocate, FoxPro, formulare, orientare pe obiect, client-server, Web
III-13423	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Marin Fotache, Ioan Brava, Cătălin Strâmbei, Liviu Creţu	Polirom	Iaşi	2002	baze de date, SQL, proceduri stocate, FoxPro, formulare, orientare pe obiect, client-server, Web
III-10678	973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Marin Fotache	Polirom	Iaşi	2001	baze de date, algebră relaţională, SQL
III-10679	973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Marin Fotache	Polirom	Iaşi	2001	baze de date, algebră relaţională, SQL

Figura 3.9. Relaţia BIBLIOTECĂ cu ordinea atributelor schimbată

După calapodul aplicat ceva mai sus, există două grupuri repetitive, `Autori` şi `CuvinteCheie`. Drept care ne grăbim să descompunem relaţia universală în trei relaţii aflate în 1NF, ca în figura 3.10.

Teoretic, baza de date din figura 3.10 este în 1NF. Schema sa diferă sensibil însă de aceeaşi BD care în 1NF arată ca în figura 3.8. Intuim că autorii şi cuvintele se referă la o carte (`ISBN`), deci la toate exemplarele sale, aşadar suntem înclinaţi să alegem varianta din figura 3.8. Atunci când numărul atributelor este mare şi situaţia reflectată este complexă, identificarea grupurilor repetitive poate deveni, pe alocuri, o loterie.

Pentru remedierea problemelor cauzate de caracterul arbitrar al dispunerii atributelor în relaţie, am putea institui o regulă: se începe cu atributele cele mai generale („non-repetitive") şi se continuă apoi către cele mai „repetitive". În cazul relaţiei BIBLIOTECĂ, s-ar putea începe cu `ISBN`-ul, continuând cu `Titlu`, `Editură`, `LocSediuEd`, `AnApariţie`, apoi `Autori` şi `CuvinteCheie`.

Similar, putem să descompunem relaţia universală FILMOGRAFIE pe baza următoarelor grupuri repetitive:

- Producători (atributul `Producător`);
- Regizori (atributul `Regizor`);
- Distribuţie (atributele `Rol` şi `Actor`);
- Genuri (atributul `Gen`);
- Premii (atributele `DenPremiu`, `LocDecernare`, `Categorie`, `AnPremiu`, `Actor`).

COTE_AUTORI

Cotă	Autor
III-13421	Marin Fotache
III-13421	Ioan Brava
III-13421	Cătălin Strâmbei
III-13421	Liviu Crețu
III-13422	Marin Fotache
III-13422	Ioan Brava
III-13422	Cătălin Strâmbei
III-13422	Liviu Crețu
III-13423	Marin Fotache
III-13423	Ioan Brava
III-13423	Cătălin Strâmbei
III-13423	Liviu Crețu
III-10678	Marin Fotache
III-10679	Marin Fotache

COTE_CUVCHEIE

Cotă	CuvântCheie
III-13421	baze de date
III-13421	SQL
III-13421	proceduri stocate
III-13421	FoxPro
III-13421	formulare
III-13421	orientare pe obiect
III-13421	client-server
III-13421	Web
III-13422	baze de date
III-13422	SQL
III-13422	proceduri stocate
III-13422	FoxPro
III-13422	formulare
III-13422	orientare pe obiect
III-13422	client-server
III-13422	Web
III-13423	baze de date
III-13423	SQL
III-13423	proceduri stocate
III-13423	FoxPro
III-13423	formulare
III-13423	orientare pe obiect
III-13423	client-server
III-13423	Web
III-10678	baze de date
III-10678	algebră relațională
III-10678	SQL
III-10679	baze de date
III-10679	algebră relațională
III-10679	SQL

CĂRȚI2

Cota	ISBN	Titlu	Editura	LocSediuEd	AnApariție
III-13421	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	Polirom	Iași	2002
III-13422	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	Polirom	Iași	2002
III-13423	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicațiilor profesionale	Polirom	Iași	2002
III-10678	973-683-709-2	SQL. Dialecte DB2, Oracle și Visual FoxPro	Polirom	Iași	2001
III-10679	973-683-709-2	SQL. Dialecte DB2, Oracle și Visual FoxPro	Polirom	Iași	2001

Figura 3.10. Prima formă normalizată a relației BIBLIOTECĂ_2

Obținem șase tabele corespunzătoare celor cinci grupuri repetitive, plus ce rămâne din relația inițială (FILME) – vezi figura 3.11.

FILME

IdFilm	TitluOriginal	TitluRO	AnLans
11899	As Good As It Gets	Mai bine nu se poate	1997
12345	Bicentennial Man	Omul bicentenar	1999

PRODUCĂTORI

IdFilm	Producător
11899	James Brooks
11899	Bridget Johnson
11899	Kristi Zea
12345	Michael Barnathan
12345	Chris Columbus
12345	Gail Katz

REGIZORI

IdFilm	Regizor
11899	James Brooks
12345	Chris Columbus

DISTRIBUŢIE

IdFilm	Rol	Actor
11899	Levin Udall	Jack Nicholson
11899	Carol Connelly	Helen Hunt
11899	Simon Bishop	Greg Kinnear
11899	Frank Sachs	Cuba Gooding Jr.
12345	Andrew Martin	Robin Williams
12345	Little Miss Amanda Martia	Embeth Davidtz
12345	Portia Charney	Embeth Davidtz
12345	Richard Martin	Sam Neil
12345	Rupert Burns	Oliver Platt
12345	Galatea	Kiersten Warren

GENURI

IdFilm	Gen
11899	comedie
11899	dramă
11899	romantic
12345	SF
12345	dramă
12345	romantic

PREMII

IdFilm	DenPremiu	LocDe-cernare	Categorie	AnPre-miu	Actor
11899	Oscar	Hollywood	cel mai bun actor în rol principal	1998	Jack Nicholson
11899	Oscar	Hollywood	cea mai bună actriţă în rol principal	1998	Helen Hunt
11899	Globul de aur	New York	cea mai bună imagine	1998	NULL
11899	Globul de aur	New York	cel mai bun actor într-o comedie/musical	1998	Jack Nicholson
11899	Globul de aur	New York	cea mai bună actriţă într-o comedie/musical	1998	Helen Hunt

Figura 3.11. Constituirea de relaţii separate pentru fiecare grup repetitiv pentru BD FILMOGRAFIE

Găselniţa funcţionează şi în cazul relaţiei referitoare la cărţile din biblioteca FEAA, şi la filmografie. Schimbăm însă datele problemei. Luăm în discuţie tabela DOTARE_JUCĂRII din figura 3.12, care gestionează „dotarea" cu jucării a fiecărui copil al unei familii. În spiritul dreptului inviolabil la proprietate şi al luptei contra colectivismului, fiecare jucărie aparţine, cel puţin oficial, unui singur copil al familiei, eventualele împrumuturi fiind rodul exclusiv al negocierilor (şi violenţei juvenile).

DOTARE_JUCĂRII

CodFamilie	NumeFamilie	Copii	Jucării	DataCumpărării
1111	Popescu Ioan	Marc	Trenuleț electric	14-05-1998
		Loredana	Păpușă Barbie	15-09-1998
		Elvis	Puzzle 200 piese	23-12-1998
			Puzzle 250 piese	14-05-1999
			Puzzle 200 piese	24-12-1999
			Căsuța Lego	08-03-2000
...				

Figura 3.12. Relație nenormalizată

Din examinarea relației de mai sus reiese că există două grupuri repetitive: `Copii` și (`Jucării`, `DataCumpărării`). Prin aplicarea regulii de separare a grupurilor repetitive obținem trei relații, COPII, JUCĂRII, FAMILII – vezi figura 3.13.

COPII

CodFamilie	Copii
1111	Marc
1111	Loredana
1111	Elvis

JUCĂRII

CodFamilie	Jucării	DataCumpărării
1111	Trenuleț electric	14-05-1998
1111	Păpușă Barbie	15-09-1998
1111	Puzzle 200 piese	23-12-1998
1111	Puzzle 250 piese	14-05-1999
1111	Puzzle 200 piese	24-12-1999
1111	Căsuța Lego	08-03-2000

FAMILII

CodFamilie	NumeFamilie
1111	Popescu Ioan

Figura 3.13. Baza de date DOTARE_JUCĂRII adusă în 1NF prin eliminarea grupurilor repetitive

Deși nu am făcut decât să urmăm un algoritm prescris ca sigur, avem de-a face cu un caz tipic de descompunere cu pierdere de informații. În cazul nostru, cele trei relații nu furnizează o informație esențială: cărui copil îi aparține o jucărie anume.

Oricum am proceda, eliminarea grupurilor repetitive nu rezolvă corect întotdeauna problema normalizării (aducerea relației în 1NF și continuarea cu 2NF ș.a.m.d.), deoarece identificarea lor se face prin observație și nu pe baza unei analize semantice riguroase a relațiilor dintre atribute.

Deși atrage un grad de redundanță considerabil, este recomandabil ca în această situație 1NF a relației să fie obținută prin introducerea în tabelă a unei linii pentru fiecare jucărie și duplicarea valorilor celorlate atribute, ca în figura 3.14.

Esențială în rezolvarea problemei este conștientizarea entității căreia i se alocă o linie în tabelă. În cadrul relației DOTARE_JUCĂRII_1, fiecare linie se referă la o jucărie afectată unui copil al unei familii.

Nici relația universală VÂNZĂRI, ale cărei atribute sunt cele din tabelul 2.1, nu este scutită de probleme la identificarea atributelor repetitive, dar amânăm discuția pentru paragraful următor.

DOTARE_JUCĂRII_1

CodFamilie	NumeFamilie	Copil	Jucărie	DataCumpărării
1111	Popescu Ioan	Marc	Trenuleț electric	14-05-1998
1111	Popescu Ioan	Loredana	Păpușă Barbie	15-09-1998
1111	Popescu Ioan	Elvis	Puzzle 200 piese	23-12-1998
1111	Popescu Ioan	Loredana	Puzzle 250 piese	14-05-1999
1111	Popescu Ioan	Marc	Puzzle 200 piese	24-12-1999
1111	Popescu Ioan	Loredana	Căsuța Lego	08-03-2000

Figura 3.14. Relația adusă în 1NF

3.4. Prima formă normală şi problema cheii primare a relaţiei universale

De cele mai multe ori, constituirea unei relaţii universale atotcuprinzătoare se loveşte de o problemă majoră - cea a cheii primare. Asupra acestui necaz, câţiva autori au atras atenţia cu ceva timp în urmă[13]. Modelul relaţional prezintă două restricţii esenţiale vizavi de cheia primară: unicitate şi valori nenule pentru atributele din alcătuirea cheii (restricţia de entitate). Or, dacă dorim să identificăm fiecare linie a relaţiei, trebuie să definim, în cele mai multe situaţii, o cheie primară cu mai multe atribute, care însă în anumite tupluri ar avea valori nule. Pe de altă parte, dacă nu vrem să fie violată restricţia de entitate, trebuie să renunţăm la a prelua în relaţia universală linii vitale din punctul de vedere al conţinutului informaţional al bazei de date.

Să luăm în discuţie câteva cazuri.

Vechea codificare poştală, valabilă până în mai 2003

Până în luna mai 2003, codurile poştale erau atribuite la nivel de oraş şi comună (cu unele excepţii) ca în figura 3.15. Dacă într-o comună erau mai multe sate, era posibil ca unele dintre acestea să partajeze acelaşi cod poştal. Singurul oraş pentru care existau mai multe coduri era Bucureştiul.

LOCALITĂŢI_CODURI_VECHI

CodPoştal	OraşComună	Sat	Judeţ
5319	Vînători	Vînători	Vrancea
5319	Vînători	Jorăşti	Vrancea
5319	Vînători	Mirceştii-Vechi	Vrancea
5613	Roznov	Roznov	Neamţ
5613	Roznov	Slobozia	Neamţ
5300	Focşani	NULL	Vrancea

Figura 3.15. Vechile coduri poştale – tabela nu este în 1NF

O localitate desemnează un oraş sau un sat. Ei bine, cheia primară ar trebui să fie combinaţia (`CodPoştal`, `OraşComună`, `Sat`). Dar valoarea atributului `Sat` este

13. Vezi, spre exemplu, [Kent 81].

NULL atunci când linia respectivă se referă la un oraş. Prin urmare, relaţia de mai sus nu se află în prima formă normală, deoarece încalcă restricţia de entitate (un atribut din cheie prezintă valori nule). Practic, pentru a fi în 1NF, relaţia nu trebuie să conţină oraşele, ceea ce este de neacceptat. Aducerea sa în 1NF se realizează prin intermediul dependenţelor funcţionale. Este drept că putem păcăli teoria şi SGBD-urile actuale definind pentru atributul `Sat` o valoare implicită nenulă – spre exemplu, un spaţiu. Aceasta miroase însă a cârpeală mioritică şi vom vedea în paragraful următor că avem soluţii mai onorabile.

Noua codificare poştală, valabilă după mai 2003

Sistemul de codificare poştală introdus în mai 2003 este mult mai fin, în sensul că un cod are acum şase cifre, în loc de patru, iar alocarea se mai face la nivel de localitate doar în cazul satelor şi oraşelor mai mici; în rest, la nivel de stradă sau chiar imobil – vezi figura 3.16.

Astfel, în tabela CODURI_NOI_V1, prima linie indică faptul că localitatea Vînători din judeţul Vrancea are codul poştal 627395, localitatea fiind şi „capitală" de comună, cel puţin după nume. Următoarele două linii se referă la alte două sate din aceeaşi comună. Valorile NULL ale atributelor `Strada` şi `Numere` pe aceste prime trei linii semnalizează faptul că pentru aceste sate nu se operează o delimitare a codurilor pe străzi (de fapt, uliţe).

Lucrurile devin cu adevărat interesante o dată cu liniile 4-7, care conţin coduri poştale ale unor adrese (locaţii, în limbaj contemporan) din municipiul Iaşi. Pentru simplificare, toate cele patru coduri sunt atribuite unor adrese aflate pe bulevardul Independenţei. Astfel, codul 700106 este asociat tuturor imobilelor din *Bd. Independenţei*, iar valoarea *1-5*, *9-13* pentru atributul `Numere` desemnează următoarele numere: 1, 3, 5 (1-5) şi 9, 11, 13 (9-13). Valoarea aceluiaşi atribut pe linia următoare (codul 700099), *18-T*, indică un set de valori pare ce începe cu 18 şi continuă cu 20, 22 etc., până la ultima adresă pară de pe bulevardul Independenţei. *T*-ul înseamnă deci „Terminare", epuizare, şi asigură atribuirea automată a acestui cod poştal eventualelor viitoare construcţii ridicate pe acest bulevard (dacă o mai fi loc). Pe linia de mai jos, valoarea *25-T* se referă la toate numerele impare de la 25 până la ultimul imobil situat pe acea parte („impară") a bulevardului. Penultima linie a tabelei semnalează un lucru oarecum deranjant: un cod poştal poate fi alocat chiar şi pentru două sau mai multe străzi, în cazul nostru, numerelor pare de la 2 la 26 şi numărului 28 de pe bulevardul Carol I, dar şi tuturor numerelor de pe aleea Veronica Micle (în tabelă este preluată titulatura de pe *www.posta-romana.ro*).

Firește, ne punem întrebarea: de ce codul 700505 nu a fost atribuit succesiunii *2-28* de pe *Bd. Carol I*, ci avem două intervale, *2-26* şi *28-28*? Răspunsul ni-l oferă rândul următor, din care aflăm că *26A* de pe *Bd. Carol I* are alt cod – 700482! Prin urmare, acelaşi număr poate fi alocat codurilor diferite, diferenţierea făcându-se prin litere sau binecunoscutul *bis*.

Cheia primară este atributul `CodPoştal`, însă relaţia este departe de 1NF. Avem de a face cu două atribute multivaloare: `Strada` şi `Numere`. O primă idee ar fi să rupem relaţia, construind câte o tabelă pentru fiecare grup repetitiv, ca în figura 3.17.

CODURI_NOI_V1

CodPoştal	Localitate	Strada	Numere	Comuna	Judeţ
627395	Vînători	NULL	NULL	Vînători	Vrancea
627397	Jorăşti	NULL	NULL	Vînători	Vrancea
627399	Mirceştii-Vechi	NULL	NULL	Vînători	Vrancea
700106	Iaşi	Bd. Independenţei	1-5 9-13	NULL	Iaşi
700099	Iaşi	Bd. Independenţei	18-T	NULL	Iaşi
700102	Iaşi	Bd. Independenţei	25-T	NULL	Iaşi
700100	Iaşi	Bd. Independenţei	7-13	NULL	Iaşi
700505	Iaşi	Bd. Carol I Aleea Micle Veronica	2-26 28 T	NULL	Iaşi
700482	Iaşi	Bd. Carol I	26A 30-32 36	NULL	Iaşi
700504	Iaşi	Bd. Carol I	28A 34 38 42	NULL	Iaşi
707295	Mircești	NULL	NULL	Mircești	Iaşi

Figura 3.16. Noile coduri poştale – tabelă nenormalizată

CODURI_LOCALITĂŢI

CodPoştal	Localitate	Comuna	Judeţ
627395	Vînători	Vînători	Vrancea
627397	Jorăşti	Vînători	Vrancea
627399	Mirceştii-Vechi	Vînători	Vrancea
700106	Iaşi	NULL	Iaşi
700099	Iaşi	NULL	Iaşi
700102	Iaşi	NULL	Iaşi
700100	Iaşi	NULL	Iaşi
700505	Iaşi	NULL	Iaşi
700482	Iaşi	NULL	Iaşi
700504	Iaşi	NULL	Iaşi
707295	Mircești	Mircești	Iaşi

CODURI_STRĂZI

CodPoştal	Strada
700106	Bd. Independenţei
700099	Bd. Independenţei
700102	Bd. Independenţei
700100	Bd. Independenţei
700505	Bd. Carol I
700505	Aleea Micle Veronica
700482	Bd. Carol I
700504	Bd. Carol I

CODURI_NUMERE

CodPoştal	Numere
700106	1-5
700106	9-13
700099	18-T
700102	25-T
700100	7-13
700505	2-26
700505	28
700505	T
700482	26A
700482	30-32
700482	36
700504	28A
700504	34
700504	38
700504	42

Figura 3.17. Tentativă eşuată de normalizare a relaţiei CODURI_NOI_V1

După cum am mai păţit-o în paragraful anterior, ruperea relaţiei iniţiale s-a făcut cu pierderi de informaţii. Deoarece un cod poştal poate fi alocat la două străzi diferite (vezi 700505), numerele din CODURI_NUMERE nu pot fi alocate corect întotdeauna.

O altă cale ar fi să înlocuim atributul `Numere` cu nu mai puţin de patru câmpuri, pentru a câştiga în limpezime informaţională, după cum urmează:

- `TipNr` (tip număr), care să conţină doar valorile:
  - NULL – pentru localităţi fără o delimitare oficială pe străzi;
  - „pare" – adică 1, 3, 5, ...;
  - „impare" – adică 2, 4, 6;
  - „ambele" – adică numere consecutive;
- `NrIniţial` (numărul iniţial) indică numărul limită inferioară a intervalului;
- `LitIniţială` (litera iniţială, dacă este cazul) se foloseşte pentru a indica dacă intervalul porneşte de la *11B* sau *24A* etc.;
- `NrFinal` (numărul final din interval) indică numărul limită superioară a intervalului;
- `LitFinală` (litera finală) semnalizează că intervalul se opreşte la *11B* sau *24A* etc.

Astfel, tabela de mai sus ar prelua forma din figura 3.18.

Spre deosebire de forma nenormalizată, tabela de mai sus ar permite găsirea imediată a codului poştal al unei adrese de genul *Bd. Carol I nr. 29*. Deşi am putea folosi un limbaj mai abstract, de genul celui din ediţia a opta a lucrării lui Chris Date, vom recurge la SQL:

```
SELECT codpostal
FROM CODURI_NOI_V2
WHERE judeţ = 'Iaşi' AND localitate='Iaşi' AND Strada='Bd.
Carol I' AND
(
(nriniţial = 29 and litiniţială IS NULL) OR
(nrfinal = 29 and litfinală IS NULL)
OR
(tipnr <> 'pare' AND nriniţial < 29 AND nrfinal > 29 )
)
```

Ca exerciţiu pentru acasă, încercaţi să aflaţi aceeaşi informaţie din schema nenormalizată a tabelei (CODURI_NOI_V2).

În modelul relaţional, este obligatoriu ca orice relaţie să posede cheie primară, alcătuită, la limită, din toate atributele relaţiei. Altfel spus, dacă în SQL o tabelă poate avea linii identice, în teoria relaţională – nu. Este una dintre „îmbunătăţirile" SQL care au atras mânia multor autori majori în domeniul bazelor de date relaţionale, precum E.F. Codd, C.J. Date, H. Darwen, F. Pascal.

CODURI_NOI_V2

CodPoştal	Localitate	Strada	TipNr	NrIniţial	LitIniţială	NrFinal	LitFinală	Comuna	Judeţ
627395	Vînători	NULL	NULL	NULL	NULL	NULL	NULL	Vînători	Vrancea
627397	Jorăşti	NULL	NULL	NULL	NULL	NULL	NULL	Vînători	Vrancea
627399	Mirceştii-Vechi	NULL	NULL	NULL	NULL	NULL	NULL	Vînători	Vrancea
700106	Iaşi	Bd. Independenţei	impare	1	NULL	5	NULL	NULL	Iaşi
700106	Iaşi	Bd. Independenţei	impare	9	NULL	13	NULL	NULL	Iaşi
700099	Iaşi	Bd. Independenţei	pare	18	NULL	NULL	NULL	NULL	Iaşi
700102	Iaşi	Bd. Independenţei	impare	25	NULL	NULL	NULL	NULL	Iaşi
700100	Iaşi	Bd. Independenţei	impar	7	NULL	13	NULL	NULL	Iaşi
700505	Iaşi	Bd. Carol I	pare	2	NULL	26	NULL	NULL	Iaşi
700505	Iaşi	Bd. Carol I	pare	28	NULL	NULL	NULL	NULL	Iaşi
700505	Iaşi	Aleea Micle Veronica	ambele	1	NULL	28	NULL	NULL	Iaşi
700482	Iaşi	Bd. Carol I	pare	26	A	NULL	NULL	NULL	Iaşi
700482	Iaşi	Bd. Carol I	pare	30	NULL	32	NULL	NULL	Iaşi
700482	Iaşi	Bd. Carol I	pare	36	NULL	NULL	NULL	NULL	Iaşi
700504	Iaşi	Bd. Carol I	pare	28	A	NULL	NULL	NULL	Iaşi
700504	Iaşi	Bd. Carol I	pare	34	NULL	36	NULL	NULL	Iaşi
700504	Iaşi	Bd. Carol I	pare	38	NULL	42	NULL	NULL	Iaşi
700504	Iaşi	Bd. Carol I	pare	42	NULL	NULL	NULL	NULL	Iaşi
707295	Mircesti	NULL	NULL	NULL	NULL	NULL	NULL	Mircesti	Iaşi

Figura 3.18. Noile coduri poştale – tabelă normalizată, dar fără cheie primară

Ei bine, pentru a identifica un tuplu unic în relaţia de mai sus, am avea nevoie de combinaţia (`CodPoştal`, `Strada`, `NrIniţial`, `LitIniţială`). Or, în majoritatea tuplurilor tabelei de mai sus, cel puţin unul dintre acestea are valori nule. După cum observăm, problema vine din regimul diferit al localităţilor fără codificare la nivel de stradă (satele şi micile oraşe). Aşa încât vom sparge relaţia CODURI_NOI_V2 în două: CODURI_SATE şi CODURI_ORAŞE. Denumirea este discutabilă, deoarece este posibil să existe sate mai mari, în curs de a deveni oraşe, cărora să li se

repartizeze mai multe coduri. Pentru simplitate, păstrăm cele două denumiri, iar conţinutul ar fi cel din figura 3.19.

CODURI_SATE

CodPoştal	Localitate	Comuna	Judeţ
627395	Vînători	Vînători	Vrancea
627397	Jorăşti	Vînători	Vrancea
627399	Mirceştii-Vechi	Vînători	Vrancea
707295	Mirceşti	Mirceşti	Iaşi

CODURI_ORAŞE

Cod Poştal	Loca-lităte	Strada	TipNr	NrIni-ţial	LitIni-ţială	NrFi-nal	LitFi-nală	Ju-deţ
700106	Iaşi	Bd. Independenţei	impare	1		5	NULL	Iaşi
700106	Iaşi	Bd. Independenţei	impare	9		13	NULL	Iaşi
700099	Iaşi	Bd. Independenţei	pare	18		NULL	NULL	Iaşi
700102	Iaşi	Bd. Independenţei	impare	25		NULL	NULL	Iaşi
700100	Iaşi	Bd. Independenţei	impar	7		13	NULL	Iaşi
700505	Iaşi	Bd. Carol I	pare	2		26	NULL	Iaşi
700505	Iaşi	Bd. Carol I	pare	28		28	NULL	Iaşi
700505	Iaşi	Aleea Micle Veronica	ambele	1		NULL	NULL	Iaşi
700482	Iaşi	Bd. Carol I	pare	26	A	26	A	Iaşi
700482	Iaşi	Bd. Carol I	pare	30		32	NULL	Iaşi
700482	Iaşi	Bd. Carol I	pare	36		36	NULL	Iaşi
700504	Iaşi	Bd. Carol I	pare	28	A	28	A	Iaşi
700504	Iaşi	Bd. Carol I	pare	34		34	NULL	Iaşi
700504	Iaşi	Bd. Carol I	pare	38		38	NULL	Iaşi
700504	Iaşi	Bd. Carol I	pare	42		42	NULL	Iaşi

Figura 3.19. Tabele separate pentru localităţile cu un singur cod şi pentru localităţile cu mai multe coduri

CODURI_SATE are o mândreţe de cheie primară: `CodPoştal`. La CODURI_ORAŞE n-am scăpat de improvizaţii. Astfel, pentru a evita nulitatea literei iniţiale, am folosit spaţiul, aşa încât cheia primară este (`CodPoştal`, `Strada`, `NrIniţial`, `LitIniţială`).

Am rezolvat o problemă şi am dat de alta: la majoritatea căutărilor trebuie să efectuăm în prealabil reuniunea celor două tabele, ceea ce e destul de apăsător. Spre exemplu, pentru primul şi ultimul cod poştal alocate judeţului Iaşi este necesară o interogare de genul:

```
SELECT MAX(codpoştal) AS cod_minim,
MAX(codpoştal) AS cod_maxim
FROM
   (SELECT codpoştal, localitate, NULL AS strada,
NULL AS tipnr, NULL AS nriniţial,
NULL AS litiniţială, NULL AS nrfinal,
NULL AS litfinală, comuna, judeţ
FROM coduri_sate
```

```
UNION
SELECT codpoştal, localitate, stradă, tipnr, nriniţial,
 litiniţială, nrfinal, litfinală, NULL, judeţu
FROM coduri_oraşe)
WHERE judet='Iaşi'
```

Marea majoritate a practicienilor preferă soluţia *gordiană*, adică folosirea unei chei surogat care este unică pentru fiecare linie (atributele de tip cheie surogat pot

CODURI_NOI_V3

Id	CodPoştal	Localitate	Strada	TipNr	NrIniţial
3456789	627395	Vînători	NULL	NULL	NULL
3456790	627397	Jorăşti	NULL	NULL	NULL
3456791	627399	Mirceştii-Vechi	NULL	NULL	NULL
3456792	700106	Iaşi	Bd. Independenţei	impare	1
3456793	700106	Iaşi	Bd. Independenţei	impare	9
3456794	700099	Iaşi	Bd. Independenţei	pare	18
3456795	700102	Iaşi	Bd. Independenţei	impare	25
3456796	700100	Iaşi	Bd. Independenţei	impar	7
3456797	700505	Iaşi	Bd. Carol I	pare	2
3456798	700505	Iaşi	Bd. Carol I	pare	28
3456799	700505	Iaşi	Aleea Micle Veronica	ambele	1
3456800	700482	Iaşi	Bd. Carol I	pare	26
3456801	700482	Iaşi	Bd. Carol I	pare	30
3456802	700482	Iaşi	Bd. Carol I	pare	36
3456803	700504	Iaşi	Bd. Carol I	pare	28
3456804	700504	Iaşi	Bd. Carol I	pare	34
3456805	700504	Iaşi	Bd. Carol I	pare	38
3456806	700504	Iaşi	Bd. Carol I	pare	42
3456807	707295	Mircești	NULL	NULL	NULL

LitIniţială	NrFinal	LitFinală	Comuna	Judeţ
NULL	NULL	NULL	Vînători	Vrancea
NULL	NULL	NULL	Vînători	Vrancea
NULL	NULL	NULL	Vînători	Vrancea
NULL	5	NULL	NULL	Iaşi
NULL	13	NULL	NULL	Iaşi
NULL	NULL	NULL	NULL	Iaşi
NULL	NULL	NULL	NULL	Iaşi
NULL	13	NULL	NULL	Iaşi
NULL	26	NULL	NULL	Iaşi
NULL	28	NULL	NULL	Iaşi
NULL	NULL	NULL	NULL	Iaşi
A	26	A	NULL	Iaşi
NULL	32	NULL	NULL	Iaşi
NULL	36	NULL	NULL	Iaşi
A	28	A	NULL	Iaşi
NULL	34	NULL	NULL	Iaşi
NULL	38	NULL	NULL	Iaşi
NULL	42	NULL	NULL	Iaşi
NULL	NULL	NULL	Mircești	Iaşi

Figura 3.20. Noile coduri poştale – tabelă normalizată, dar fără cheie primară

fi gestionate automat în majoritatea SGBD-urilor) - vezi tabela CODURI_NOI_V3 din figura 3.20. Risipa de spaţiu generată de introducerea acestui atribut reprezintă un preţ rezonabil la durerile de cap pricinuite de valorile nule ale „fostelor" atribute chei. Atributul `Id` nu are nici o relevanţă informaţională, ci doar ne ajută să asigurăm unicitatea cheii primare fără pericolul încălcării restricţiei de entitate.

Atenţie, însă! Cheile surogat constituie o tentaţie căreia (ca oricărei tentaţii) trebuie să i se dea curs cu moderaţie.

Baza de date VÂNZĂRI

Şi relaţia universală a bazei de date VÂNZĂRI „suferă" de aceeaşi problemă a cheii primare, şi, implicit, prima formă normală este mai mult decât discutabilă. Conform „preceptelor" relaţionale, VÂNZĂRI {Jud, Judeţ, Regiune, CodPost, Localitate, Comună, CodCl, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPr, DenPr, UM, Grupa, ProcTVA, NrFact, DataFact, Obs, Linie, Cantitate, PreţUnit, CodÎnc, DataÎnc, CodDoc, NrDoc, DataDoc, Tranşă} trebuie să prezinte o combinaţie de atribute ale cărei valori conferă unicitate fiecărui tuplu.

Ţinând seama că:

- un client are un singur sediu central (poate avea filiale, însă ne limităm la a considera că facturarea se face pe adresa sediului central);
- o factură emisă are un număr unic şi se întocmeşte unui singur client;
- o factură conţine una sau mai multe linii;
- pe fiecare linie este înregistrat un produs, fiecare vândut într-o anumită cantitate şi la un anumit preţ unitar;
- deoarece este posibilă rediferenţierea procentului de taxă pe valoarea adăugată în funcţie de categoria produselor (alimentare, medicamente, cărţi, bunuri considerate de lux etc.), `ProcTVA` este bine de asociat fiecărui produs;
- o încasare are la bază un document justificativ;
- o factură poate fi plătită de client în tranşe;
- la o încasare se pot achita, integral sau parţial, una sau mai multe facturi; cheia primară ar fi combinaţia (`NrFact`, `Linie`, `CodÎnc`). Necazul este că la momentul întocmirii nu se cunosc datele despre încasări. Mai mult, acestea pot surveni la intervale mari de timp, mai ales în condiţiile blocajelor financiare care ne-au făcut celebri în mediul economic european.

Potrivit primei soluţii care nu încalcă restricţia de entitate, inserarea de linii în tabela VÂNZĂRI se face abia în momentul încasării facturii, ceea ce este inadmisibil, întrucât baza de date va fi incapabilă să furnizeze informaţii vitale precum:

- vânzările pe ziua, săptămâna, luna, anul curente;
- vânzările pe produse;
- vânzările pe clienţi;
- creanţele (valoarea rămasă de încasat) faţă de clienţi.

A doua soluţie este spargerea relaţiei universale. Cum? Prin eliminarea grupurilor repetitive. Dar care sunt grupurile repetitive, în condiţiile în care o factură poate fi achitată în tranşe (mai multe încasări), iar o încasare poate „regla" mai multe

facturi? Vom vedea în capitolele viitoare cum, pe baza dependenţelor funcţionale, putem obţine o structură mult mai acceptabilă.

Baza de date FILMOGRAFIE

În paragraful anterior am obţinut o schemă ce se apropie destul de mult de rezonabil, cel puţin în materie de eliminare a grupurilor repetitive. Dintre cele şase tabele ale bazei de date FILMOGRAFIE din figura 3.11, să examinăm preţ de câteva minute pe ultima - PREMII. Cheia primară este combinaţia (`IdFilm`, `DenPremiu`, `Categorie`, `AnPremiu`), în condiţiile în care orice premiu de interpretare se acordă unui singur actor/actriţă. Ce s-ar fi întâmplat însă dacă *Oscarul pentru cel mai bun actor în rol principal* ar fi fost acordat simultan lui Robert de Niro şi Al Pacino pentru prestaţia lor dintr-un film celebru, în care, practic, nu putem spune că unul dintre roluri este principal, iar celălalt secundar. Toate cele patru atribute ale cheii ar prezenta valori identice pe două tupluri (un tuplu pentru de Niro, celălalt pentru Pacino), iar restricţia de cheie primară ar fi încălcată.

Când proiectăm baze de date, trebuie să ne gândim şi la astfel de speţe marginale ce pot compromite o structură chiar după ani buni de funcţionare a aplicaţiei. Fiind tentaţi să adăugăm celor patru atribute din cheie pe al cincilea, `Actor`, nu putem să nu observăm că, pentru premiile acordate la categoriile regie, scenariu etc., acest atribut are valoarea NULL, ceea ce violează restricţia de entitate.

Soluţia cea mai la îndemână ţine de ruperea acestei relaţii în două, una dedicată premiilor de interpretare, iar a doua celorlalte premii - vezi figura 3.21.

PREMII_INTERPRETARE

IdFilm	DenPremiu	LocDecernare	Categorie	AnPremiu	Actor
11899	Oscar	Hollywood	cel mai bun actor în rol principal	1998	Jack Nicholson
11899	Oscar	Hollywood	cea mai bună actriţă în rol principal	1998	Helen Hunt
11899	Globul de aur	New York	cel mai bun actor într-o comedie/musical	1998	Jack Nicholson
11899	Globul de aur	New York	cea mai bună actriţă într-o comedie/musical	1998	Helen Hunt

CELELALTE_PREMII

IdFilm	DenPremiu	LocDecernare	Categorie	AnPremiu
11899	Globul de aur	New York	cea mai bună imagine	1998

Figura 3.21. Ruperea relaţiei PREMII din cauza problemelor de cheie primară

3.5. Dezbateri pe marginea şi în centrul primei forme normale

Prima formă normală este, fără îndoială, un ciudat amestec de facil şi confuz. Putem spune chiar că mare parte din doza de facil, de superficialitate din cursurile universitare, cărţile dedicate bazelor de date sau chiar proiectării bazelor de date (inclusiv normalizării) îşi are sorgintea tocmai în confuzia care a învăluit, încă de la începuturile relaţionalului, definirea primei forme normale şi atomicităţii[14]. Unii autori, precum Elmasri şi Navathe, consideră restricţia de atomicitate proprie modelului relaţional, restricţie care este eliminată din modelele relaţional imbricat (*nested relational model*) şi din cel obiectual-relaţional care, ambele, permit relaţii nenormalizate[15].

Ceea ce se înţelege îndeobşte prin atomicitatea atributelor ţine de caracterul scalar al valorilor: întregi, şiruri de caractere, date calendaristice. Prin comparaţie, un atribut non-atomic este unul definit pe un domeniu de valori compozite, complexe. Fiecare componentă dintr-o valoare compozită poate fi, la rândul ei, compozită, ajungându-se la o structură ierarhică şi flexibilă, în funcţie de natura obiectului sau procesului modelat. Elmasri şi Navathe sunt cât se poate de limpezi: pentru atributele unei relaţii în 1NF, singurele valori permise sunt cele atomice sau indivizibile[16].

În ceea ce mă priveşte, materialul favorit legat de 1NF este cel al lui Chris Date, publicat în iunie 2003 pe site-ul *http://www.dbdebunk.com*[17], site conceput şi gestionat de un apropiat al lui Date şi un necruţător observator al ignoranţei comunităţii IT (şi academice) în materie de baze de date – Fabian Pascal. Materialul evocat constituie deopotrivă încununarea şi convergenţa lucrărilor scrise singur (cea mai celebră este cartea a cărei a opta ediţie a fost tradusă şi în româneşte) sau împreună cu Hugh Darwen (vezi *The Third Manifesto*). Meritoriu este şi faptul că, din capul locului, Date se autodenunţă ca fiind unul dintre cei care, de-a lungul timpului, au contribuit la confuzia în care se scaldă atomicitatea.

Prima definiţie a 1NF aparţine, firește, lui E.F. Codd, pentru care o relaţie este în 1NF dacă nici unul dintre domeniile sale nu are elemente (valori) de tip set[18]. Câţiva ani mai târziu, Codd afirma că un domeniu este simplu dacă toate valorile sale sunt atomice, adică *nedecompozabile de către SGBD*[19]. Date identifică în cartea lui Codd publicată în 1990[20] două afirmaţii similare:

- valorile din domeniile pe care fiecare relaţie este definită sunt în mod necesar atomice vizavi de SGBD;

14. Există lucrări respectabile, precum [Atzeni ş.a. 99], care nici nu fac aluzie la prima formă normală sau la atomicitatea valorilor într-o relaţie.
15. [Elmasri & Navathe 00], p. 485.
16. [Elmasri & Navathe 00], p. 485.
17. [Date 03].
18. [Codd 72]. Preluare din [Date 03].
19. [Codd 79].
20. [Codd 90].

- datele atomice sunt cele care nu pot fi descompuse de către SGBD în subcomponente (cu excepţia unor funcţii speciale).

Din păcate, Codd elimină o neclaritate înlocuind-o cu alta. Şi Date se întreabă: ce înseamnă *cu excepţia unor funcţii speciale*? Faptul că dintr-un şir de caractere (exemplul clasic de valoare atomică) pot fi extrase diferite componente prin funcţii precum SUBSTR, funcţii prezente în mai toate SGBD-urile actuale, face din valoarea de tip şir de caractere una neatomică? Sună cam hilar. Mai ales că, dacă raportăm atomicitatea la SGBD-uri, deseori vom fi în situaţia în care o valoare atomică într-o bază de date gestionată cu un SGBD să fie neatomică în cazul unei aceleiaşi baze de date (structuri) implementate pe un alt SGBD. Ceea ce, să recunoaştem, nu e prea onorant pentru un model atât de riguros fundamentat aşa cum este relaţionalul.

Date, împreună cu Darwen şi Pascal propun renunţarea la atomicitate ca o cerinţă a 1NF. După Date, atomicitatea nici nu are o semnificaţie absolută, întrucât depinde de ceea ce dorim să facem cu datele asupra atomicităţii cărora ne pronunţăm[21]. Practic, toate tabelele care respectă principiile modelului relaţional sunt în 1NF, iar această primă formă normală rămâne relevantă doar în două privinţe:

- indică faptul că relaţia la care se referă poate să nu fie într-o formă normalizată superioară (2NF, 3NF...);
- o structură de date ce respectă 1NF este relaţională, cu alte cuvinte tabelele care nu sunt în 1NF sunt non-relaţionale.

Acest din urmă aspect merită o discuţie detaliată, deoarece ţine de aspectul structural al modelului relaţional.

Renunţarea la atomicitate nu este singura schimbare fundamentală în modelul relaţional. Domeniile pot conţine orice tip de valori: scalare sau compozite, şiruri de caractere, numere, dar şi vectori, liste, imagini, înregistrări audio/video, documente XML sau orice tip definit de utilizatori. Dealminteri, dacă iniţial Codd s-a străduit să impună sintagma *domeniu* în detrimentul *tipului*, tocmai pentru a opera o distrincţie netă între baze de date şi limbaje de programare, Date, Darwen şi Pascal sunt, în ultimul timp, mult mai apropiaţi de noţiunea de *tip*, pe care o consideră mai sugestivă şi, în plus, identic aplicabilă şi altor modele de organizare a datelor. Atâta vreme cât există suport din partea SGBD-ului pentru definirea, stocarea şi accesarea sa, se poate folosi orice tip de date într-o BD.

Revenim la relaţia BIBLIOTECĂ din figura 3.1. Pe baza tipului şir de caractere, putem defini un domeniu de tip *vectori de şiruri de caractere*, iar cele trei atribute, `Cote`, `Autori` şi `CuvinteCheie` se vor declara pe acest nou domeniu de tip set. Chiar dacă noua relaţie, denumită BIBLIOTECĂ_SET (figura 3.22), seamănă izbitor cu forma denormalizată, este o schemă radical diferită.

A doua inovaţie o constituie atributele ale căror valori pot fi chiar relaţii (*relation-valued attributes* – ATR). În exemplul nostru, în locul celor trei seturi putem folosi

21. [Date 03].

BIBLIOTECĂ_SET

ISBN	Titlu	Cote_SET	Autori_SET
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	{III-13421,III-13422, III-13423}	{Marin Fotache, Ioan Brava, Cătălin Strâmbei, Liviu Creţu}
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	{III-10678, III-10679}	{Marin Fotache}

Editura	LocSediuEd	AnApariţie	CuvinteCheie_SET
Polirom	Iaşi	2002	{baze de date, SQL, proceduri stocate, FoxPro, formulare, orientare pe obiecte, client-server, web}
Polirom	Iaşi	2001	{baze de date, algebră relaţională, SQL}

Figura 3.22. Domenii de tip set

câte o relaţie. Câştigul este important, deoarece ingredientul folosit este pur relaţional (relaţia!), iar mecanismul de declarare a restricţiilor şi cel de interogare (algebra relaţională) sunt, în esenţă, aceleaşi. Atributele `Cote_REL`, `Autori_REL` şi `Cuvinte Cheie_REL` din relaţia BIBLIOTECĂ_ATR (figura 3.23) sunt de acest tip, iar valorile lor pot fi supuse operatorilor clasici: selecţie, proiecţie, joncţiune etc. Date şi Harwen propun câţiva noi operatori algebrici relaţionali pentru a asigura comparabilitatea tabelelor cu şi fără ATR, GROUP şi UNGROUP[22].

BIBLIOTECĂ_ATR

ISBN	Titlu	Cote_REL
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Cote: III-13421; III-13422; III-13423
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Cote: III-10678; III-10679

Autori_REL	Editura	LocSediuEd	AnApariţie	CuvinteCheie_REL
Autori: Marin Fotache; Ioan Brava; Cătălin Strâmbei; Liviu Creţu	Polirom	Iaşi	2002	CuvinteCheie: baze de date; SQL; proceduri stocate; FoxPro; formulare; orientare pe obiect; client-server; Web
Autori: Marin Fotache	Polirom	Iaşi	2001	CuvinteCheie: baze de date; algebră relaţională; Stocate

Figura 3.23. Atribute de tip relaţii (ATR)

22. Vezi [Date & Darwen 00].

Din contră, dacă am folosi prima variantă - cea a seturilor - ar fi necesari operatori speciali: reuniune de seturi, intersecţie de seturi etc.

Aceaştă a doua inovaţie a modelului relaţional este o consecinţă directă a primei. Practic, dacă un atribut X este de tip relaţie, iar D este domeniul lui X, atunci toate valorile lui D sunt relaţii. În plus, orice atribut de tip relaţie poate conţine atribute care sunt, la rândul lor, de tip relaţii, astfel încât numărul nivelurilor de imbricare este nelimitat.

Interesantă este în acest sens şi lucrarea publicată în 1984 de către Serge Abiteboul şi Nicole Bidoit. Deşi autorii afirmau că propun un model de date nou (l-au botezat Verso), în mare ideea era ca valoarea unui atribut să fie nu numai atomică, ci şi o instanţă a unui *format* (autorii au ezitat să folosească termenul *relaţie*). Ceea ce este notabil la modelul propus este că prin acest mecanism era obţinută o ierarhie ce putea fi interogată folosind aceiaşi operatori algebrici relaţionali. Cu atât mai mult cu cât valorile nule aveau o formă de reprezentare care ar mulţumi astăzi pe Date şi compania (relaţii vide)[23].

Howard Dreizen şi Shi-Huo Chang propun în numărul din decembrie 1989 al *ACM Transactions on Database Systems* acceptarea, cu titlu restrictiv, a unor condiţii excepţionale în schema bazei de date, pentru a rezolva o serie de situaţii practice relativ rare care însă creează anomalii în schema bazei. Cu acest prilej, autorii se pronunţă pentru folosirea *relaţiilor incluse*, altfel spus, includerea relaţiilor în alte relaţii, de o manieră nerestrictivă; astfel, o relaţie R (D1, D2, ..., Dn) de *n* domenii este recursivă omogen dacă fiecare domeniu este fie (1) un set de valori atomice, fie (2) un set de relaţii recursive omogen cu scheme identice[24].

O dată trecută euforia contactului cu atributele de tip relaţii, ne putem întreba retoric, precum Date: în proiectarea schemei bazei chiar avem nevoie de ATR-uri? Întrebarea corectă ar fi, mai degrabă: când sunt ATR-urile mai avantajoase, prin comparaţie cu cele atomice (şi mecanismul de normalizare expus în acest capitol)? Principalul avantaj ţine, după cum am discutat, de enorma lor flexibilitate. Cât despre limite sau, după caz, dezavantaje, ar merita început cu structura ierarhică a ATR, ca şi în lumea orientării pe obiect, în general. Nu întotdeauna realitatea este ierarhică. Relaţiile dintre obiecte pot fi cu mult mai complexe, iar de cele mai multe ori sunt necesare regrupări, resistematizări ale datelor, situaţii în care stângăciile structurilor ierarhice se manifestă în toată splendoarea lor.

Să luăm în discuţie relaţia STUDENŢI_EXAMENE din figura 2.3, pentru care încercăm să valorificăm ATR-urile. Prima variantă - vezi figura 3.24 - foloseşte un atribut de tip relaţie, numit `Examene_ATR`, ce conţine informaţii despre toate examenele susţinute de un student. Relaţia STUDENŢI_EXAMENE_ATR1 va conţine doar câte un tuplu pentru fiecare student.

În egală măsură, se poate însă folosi şi un ATR care să conţină toţi examinaţii pentru o disciplină dată într-o sesiune - `StudenţiNote_ATR`. Relaţia s-ar prezenta după cum este sugerat în figura 3.25.

23. Vezi [Abiteboul & Bidoit 84].
24. [Dreizen & Chang 89].

STUDENȚI_EXAMENE_ATR1

Matricol	NumePrenume	An	Specializare
EL13455	Popovici I Vasile	3	Informatică economică
EL13456	Zăineanu W Ion	3	Informatică economică
EL13457	Abălaşei R Zicu	3	Informatică economică
EL13458	Şpagă M Michael	3	Informatică economică

Examene_ATR

CodDisc	DenumireDisc	NrCredite	DataExamen	Nota
AI3501	Baze de date I	6	29/01/2004	8
AI3502	Programare vizuală şi RAD	7	01/02/2004	10

CodDisc	DenumireDisc	NrCredite	DataExamen	Nota
AI3501	Baze de date I	6	29/01/2004	4
AI3502	Programare vizuală şi RAD	7	01/02/2004	8
AI3501	Baze de date I	6	12/02/2004	8

CodDisc	DenumireDisc	NrCredite	DataExamen	Nota
AI3501	Baze de date I	6	29/01/2004	9
AI3502	Programare vizuală şi RAD	7	01/02/2004	4
AI3502	Programare vizuală şi RAD	7	15/02/2004	9

CodDisc	DenumireDisc	NrCredite	DataExamen	Nota
AI3503	Analiza sistemelor informaționale	6	04/02/2004	7

Figura 3.24. Prima variantă de folosire a ATR

Care dintre cele două variante reflectă mai bine realitatea ? Prima variantă prezintă avantajul grupării tuturor examenelor unui student sub „umbrela" tuplului care se referă la studentul respectiv. Ar fi relativ simplu de calculat media studentului, de aflat la ce examene a picat o singură dată, sau de două (trei...) ori etc. Dar dacă ne interesează studenţii care au luat la Programare orientată pe obiect aceeaşi notă ca şi Şpagă Michael, atunci interogarea s-ar complica destul. A doua variantă este, ca structură, mai aproape de realitate, deoarece notele sunt preluate de pe cataloage care se întocmesc la fiecare examen. Avantajul obţinerii lejere a catalogului de la examen este „compensat" de dificultatea calculării mediilor pentru un student sau o formaţie de studiu, pentru comparaţii între situaţiile şcolare ale studenţilor etc.

Din acest punct de vedere, renunţarea la ATR şi lucrul cu atribute atomice se materializează într-o structură, să-i spunem, mai neutră, care, deşi nu e atât de intuitivă precum cea ierarhică, se pretează mult mai bine la prelucrări dintre cele mai diverse.

STUDENŢI_EXAMENE_ATR2

CodDisc	DenumireDisc	NrCredite	DataExamen
AI3501	Baze de date I	6	29/01/2004
AI3501	Baze de date I	6	12/02/2004
AI3502	Programare vizuală şi RAD	7	01/02/2004
AI3502	Programare vizuală şi RAD	7	15/02/2004
AI3503	Analiza sistemelor informaţionale	6	04/02/2004

StudenţiNote_ATR

Matricol	NumePrenume	An	Specializare	Nota
EL13455	Popovici I Vasile	3	Informatică economică	8
EL13456	Zăineanu W Ion	3	Informatică economică	4
EL13457	Abălaşei R Zicu	3	Informatică economică	9

Matricol	NumePrenume	An	Specializare	Nota
EL13456	Zăineanu W Ion	3	Informatică economică	8

Matricol	NumePrenume	An	Specializare	Nota
EL13455	Popovici I Vasile	3	Informatică economică	10
EL13456	Zăineanu W Ion	3	Informatică economică	8
EL13457	Abălaşei R Zicu	3	Informatică economică	4

Matricol	NumePrenume	An	Specializare	Nota
EL13457	Abălaşei R Zicu	3	Informatică economică	9

Matricol	NumePrenume	An	Specializare	Nota
EL13458	Spagă M Michael	3	Informatică economică	7

Figura 3.25. A doua variantă de folosire a ATR

După discuţia din acest paragraf, în care domeniile pot fi definite cât se poate de flexibil, în funcţie de nevoile aplicaţiei, ne putem pune întrebarea: ce rost mai are să discutăm despre aducerea de relaţii în 1NF, operaţiune care, uneori, presupune creşterea numărului de tupluri de câteva ori, astfel încât în loc să micşorăm redundanţa, mai degrabă o mărim, cel puţin aparent?

Două ar fi argumentele în sprijinul celor prezentate în acest capitol până în paragraful în care ne aflăm (oricum este ultimul!). Mai întâi, în procesul normalizării, acesta e doar primul pas. Redundanţa pe care o introducem rezolvă o problemă extrem de importantă, cea a pierderilor de informaţii. După cum o să vedem în capitolele următoare, este aproape sigur că în celelalte forme normalizate vom scăpa de aproape tot ce este redundant într-o relaţie.

Al doilea argument ţine de instrumentele software de care dispunem. La acest moment, suportul SGBD-urilor pentru domenii de tip set, ca să nu mai vorbim de

atribute de tip relaţie, este mai degrabă unul modest. Chiar dacă produsele importante au facilităţi deosebite în definirea de obiecte, când se pune problema mecanismului de declarare a integrităţii obiectelor, şi mai ales a celui de interogare, lucrurile se acutizează. Chiar dacă teoretic opţiunile sunt generoase, atunci când se ridică problema punerii în operă a unei aplicaţii de lucru cu baze de date nu putem eluda „meandrele concretului", cu atât mai puţin „sinergia faptelor".

CAPITOLUL 4

Dependenţe funcţionale. A doua formă normalizată

Cea de-a doua formă normală (2NF) se bazează pe dependenţele funcţionale *totale* sau *elementare*. Drept care acest capitol începe cu prezentarea dependenţelor funcţionale (DF), apoi tratează câteva modalităţi de reprezentare a DF, iar în continuare sunt puse în evidenţă elementele care împart dependenţele funcţionale în elementare (totale) şi neelementare (parţiale). În final, pe baza acestei tipologii sunt prezentate modalităţile de „rupere" a bazei de date în mai multe relaţii aflate în 2NF. În plus, după cum vom vedea, eliminarea dependenţelor funcţionale parţiale rezolvă şi problema bazelor de date a căror schemă în 1NF încalcă restricţia de entitate.

4.1. Dependenţe funcţionale

Problema stabilirii dependenţelor dintre atribute este fundamentală în procesul elaborării bazelor de date relaţionale. Aducerea structurii bazei într-o variantă care minimizează redundanţele şi anomaliile manifestate la adăugarea/modificarea/ştergerea unor linii din tabele presupune parcurgerea unui proces de normalizare a relaţiilor, normalizare care este centrată tocmai pe conceptul de dependenţă între atributele bazei.

4.1.1. *Definiţie*

Dependenţa funcţională reprezintă o generalizare a conceptului de cheie primară[1]. Desemnăm prin `Data2` un atribut sau o combinaţie de atribute dintr-o relaţie oarecare şi prin `Data1` un alt atribut sau grup de atribute din aceeaşi relaţie. Se spune că `Data2` este în *dependenţă funcţională* (sau depinde funcţional) de `Data1` atunci când cunoaşterea unei valori pentru `Data1` permite determinarea (cunoaşterea) a maximum o valoare pentru `Data2`. `Data1` se numeşte *sursa* sau *determinantul* dependenţei funcţionale, iar `Data2` reprezintă *destinaţia* (*scopul* sau *determinatul*) dependenţei funcţionale.

1. Vezi şi [Garcia-Molina ş.a. 02], [Date 86].

Formal, dacă notăm relaţia R $\{A_1, A_2, \ldots, A_n\}$ - unde R este numele relaţiei şi A_i sunt atributele relaţiei R - şi prin X şi Y două subansambluri din $\{A_1, A_2, \ldots, A_n\}$, se spune că există o dependenţă funcţională între X şi Y (simbolizată X ⟶ Y) dacă şi numai dacă:

- fiecare apariţie (valoare) a lui X poate fi asociată unei singure apariţii a lui Y;
- două apariţii identice ale lui X nu pot fi asociate decât aceleiaşi apariţii a lui Y.

Într-o prezentare şi mai explicită, Date[2] defineşte dependenţa funcţională după cum urmează: dată fiind o relaţie R, subansamblul de atribute Y din R depinde funcţional de subansamblul X (tot din R), dacă şi numai dacă ori de câte ori două tupluri din R prezintă aceeaşi valoare pentru X, obligatoriu valoarea celor două tupluri este identică şi pentru Y.

Se spune despre o dependenţă că este *trivială* dacă şi numai dacă destinaţia este un subset al sursei: dacă $Y \subseteq X$, atunci X ⟶ Y.

Revenim asupra relaţiei universale VÂNZĂRI, prezentată în capitolul 2 (tabelul 2.1) şi discutată în finalul capitolului 3 (paragraful 3.4.). Cunoaşterea numărului facturii permite determinarea cu exactitate a codului clientului, deoarece o factură este (conform legii şi metodologiei în vigoare) întocmită pentru un singur client. Se poate scrie:

```
NrFact ⟶ DataFact
NrFact ⟶ CodCl
NrFact ⟶ Obs
```

dar şi

```
NrFact ⟶ (DataFact, CodCl)
NrFact ⟶ (DataFact, CodCl, Obs)
```

Situaţia de mai sus se încadrează în cazurile în care sursa (determinantul) este cheia primară a relaţiei. Implicit, în orice relaţie R există dependenţele funcţionale:

cheia lui R ⟶ celelalte atribute ale lui R.

Luăm un alt exemplu: un cod poştal nu poate desemna adrese din oraşe şi comune diferite. Se poate scrie, deci:

```
CodPoştal ⟶ Localitate
CodPoştal ⟶ Judeţ
```

Observaţie

Uneori, începătorii au probleme în ceea ce priveşte „sensul" dependenţelor funcţionale. Dependenţa funcţională CodPoştal ⟶ Localitate nu înseamnă

2. [Date 86], p. 365.

că o localitate are un singur cod poştal, ci invers, că *unui cod poştal îi corespunde o singură localitate*.

O dependenţă funcţională în care scopul (destinaţia) este alcătuit dintr-un singur atribut se numeşte *dependenţă funcţională canonică* sau dependenţă funcţională în formă canonică[3].

Două atribute *nu* sunt în dependenţă funcţională atunci când cunoaşterea unei valori oarecare a primului atribut:

- fie nu permite cunoaşterea nici uneia dintre valorile celui de-al doilea (nu există nici un „raport" între cele două atribute); în acest caz, dependenţa nefuncţională echivalează cu inexistenţa vreunei dependenţe între cele două atribute;
- fie permite cunoaşterea mai multor valori ale celui de-al doilea atribut.

Relaţia VÂNZĂRI

Între atributele `DataFact` şi `CodCl` nu există dependenţă funcţională. O valoare a atributului `DataFact` nu furnizează nici o informaţie cu privire la valoarea de pe aceeaşi linie a atributului `CodCl`. Se scrie:

`DataFact` —/→ `CodCl`

De asemenea, deoarece pentru un client se pot întocmi mai multe facturi (aferente fiecărei vânzări):

`CodCl` —/→ `NrFact`

Într-o factură se poate consemna vânzarea mai multor produse/servicii; prin urmare, unei valori a `NrFact` îi corespund mai multe valori ale atributului `CodPr`. Se notează:

`NrFact` —/→ `CodPr`.

Relaţia LINII_ARTICOLE_CONTABILE

Să examinăm tabela din figura 4.1:

LINII_ARTICOLE_CONTABILE

NotaContabilă	Data	Operaţiune	ContDebitor	ContCreditor	Suma
150	30.11.2002	1	300	401	10000000
150	30.11.2002	1	4426	401	1800000
150	30.11.2002	2	401	5311	7000000
150	30.11.2002	2	401	5121	4800000
151	01.12.2002	1	5121	700	5250000

Figura 4.1. Tabela LINII_ARTICOLE_CONTABILE

3. [Saleh 4], p. 76. Vezi şi [Garcia-Molina ş.a. 01], p. 89.

- nota contabilă este alcătuită din mai multe articole contabile:
  (NotaContabilă, Data) —/→ Operaţiune;
- un articol contabil (înregistrare) poate fi compus, adică alcătuit, din mai multe corespondenţe ContDebitor = ContCreditor:
  (NotaContabilă, Data, Operaţiune) —/→ ContDebitor;
  (NotaContabilă, Data, Operaţiune) —/→ ContCreditor;
  (NotaContabilă, Data, Operaţiune, ContDebitor) —/→ ContCreditor;
  (NotaContabilă, Data, Operaţiune, ContCreditor) —/→ ContDebitor.
  Atunci când firmele îşi centralizează operaţiunile, utilizând de la o lună la alta aceleaşi numere pentru notele contabile:
  NotaContabilă —/→ Data.

Pe lângă aceste două exemple, am mai putea aminti faptul că localităţi cu aceeaşi denumire există în mai multe judeţe. Spre exemplu, localitatea Vînători există în judeţele Vrancea, Galaţi, Iaşi, Neamţ etc. Aceasta înseamnă că:

Localitate —/→ Judeţ

Dependenţele funcţionale reprezintă legături semantice între atribute ale bazei de date. Au fost propuşi algoritmi de identificare şi gestionare a dependenţelor funcţionale într-o relaţie. Cea mai mare parte a acestora sunt total inutili, pentru că, efectiv, nimeni nu poate stabili o dependenţă fără a cunoaşte regula sau restricţia care se ascunde în spatele acesteia, altfel spus, fără a cunoaşte temeinic realitatea modelată. Cu atât mai mult calculatorul care are mari probleme în a lucra la nivel semantic, ca să nu mai vorbim de cel pragmatic.

4.1.2. *Dependenţe funcţionale cu sursa compusă*

Există dependenţe funcţionale care prezintă în partea stângă (sursa dependenţei funcţionale sau determinantul) două sau mai multe atribute. Acestea sunt dependenţele funcţionale cu sursa compusă. Astfel, în relaţia VÂNZĂRI:

- o factură conţine mai multe linii: NrFact —/→ Linie;
- pe o factură pot apărea mai multe produse: NrFact —/→ CodPr;
- un produs apare, de obicei, pe mai multe facturi: CodPr —/→ NrFact;
- pentru fiecare factură, o linie este unică:
  (NrFact, Linie) ⟶ CodPr;
  (NrFact, Linie) ⟶ Cantitate;
  (NrFact, Linie) ⟶ PreţUnit;
- dacă stabilim că un produs nu poate apărea pe o factură decât o singură dată, atunci sunt valabile DF:
  (NrFact, CodPr) ⟶ Linie;
  (NrFact, CodPr) ⟶ Cantitate;
  (NrFact, CodPr) ⟶ PreţUnit;

Relaţia LINII_ARTICOLE_CONTABILE

În tabela din figura 4.1, stabilirea dependenţelor funcţionale depinde de modul în care firma îşi organizează evidenţa contabilă. Astfel, marea majoritate a firmelor îşi centralizează operaţiunile pe tipuri (intrări de materiale, consumuri de materiale, operaţiuni legate de casierie, conturile din bancă etc.), utilizând, de la o lună la alta, aceleaşi numere pentru notele contabile (de exemplu: nota 1 – încasări prin casierie, nota 10 – pentru înregistrarea amortizării etc.). În acest caz, există dependenţa funcţională:

(1) (NotaContabilă, Data, Operaţiune, ContDebitor, ContCreditor) ⟶ Suma

Dacă în unele firme, cu un volum de activitate mai redus, ar conta fiecare document primit/întocmit, generând astfel o nouă notă contabilă, dependenţele funcţionale s-ar putea scrie sub forma:

(2) NotaContabilă ⟶ Data

(3) (NotaContabilă, Operaţiune, ContDebitor, ContCreditor) ⟶ Suma

Firește, este valabilă şi dependenţa:

(4) (NotaContabilă, Operaţiune, ContDebitor, ContCreditor) ⟶ Data

Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3

Tabela din figura 3.7. (paragraful 3.3.2), aflată în prima formă normală, prezintă ca cheie primară combinaţia (Cota, Autor, CuvântCheie). De aceea, se pot inventaria următoarele cinci dependenţe funcţionale, în care destinaţii sunt atributele necheie:

(1) (Cota, Autor, CuvântCheie) ⟶ ISBN

(2) (Cota, Autor, CuvântCheie) ⟶ Titlu

(3) (Cota, Autor, CuvântCheie) ⟶ Editura

(4) (Cota, Autor, CuvântCheie) ⟶ LocSediuEd

(5) (Cota, Autor, CuvântCheie) ⟶ AnApariţie

Alte elemente luate în discuţie pentru stabilirea DF:

- o carte poate fi scrisă de mai mulţi autori: ISBN ⟶/ Autor;
- un autor scrie mai multe cărţi: Autor ⟶/ ISBN şi
- despre mai multe subiecte: Autor ⟶/ CuvântCheie;
- într-o carte sunt tratate mai multe subiecte: ISBN ⟶/ CuvântCheie;
- pentru o carte sunt în depozit mai multe exemplare: ISBN ⟶/ Cota.

Cota este codul care identifică în mod unic o carte aflată în depozit, aşadar:

(6) Cota ⟶ ISBN

(7) Cota ⟶ Titlu

(8) Cota ⟶ Editura

(9) `Cota` ⟶ `LocSediuEd`

(10) `Cota` ⟶ `AnApariţie`

Fiecare titlu de carte este identificat fără ambiguitate de `ISBN`:

(11) `ISBN` ⟶ `Titlu`

(12) `ISBN` ⟶ `Editura`

(13) `ISBN` ⟶ `LocSediuEd`

(14) `ISBN` ⟶ `AnApariţie`

O editură are un singur sediu central:

(15) `Editura` ⟶ `LocSediuEd`

Prin urmare, dintre cele 15 DF inventariate, primele cinci au sursa compusă, celelalte, sursa simplă.

Relaţia DOTARE_JUCĂRII_1

Pentru identificarea dependenţelor funcţionale care se manifestă în relaţia prezentată în figura 3.14 (finalul paragrafului 3.3.3), discuţia poate fi purtată după cum urmează:

- codul identifică o familie:
  (1) `CodFamilie` ⟶ `NumeFamilie`;
- o familie are mai mulţi copii: `CodFamilie` ⟶/ `Copil`;
- un copil face parte dintr-o singură familie, însă atributul `Copil` nu identifică în mod unic un singur copil, deoarece se referă numai la prenume (Ion, Vasile etc.):
  `Copil` ⟶/ `CodFamilie`;
- cu toată recesiunea, un copil are mai multe jucării:
  (`CodFamilie`, `Copil`) ⟶/ `Jucărie`;
- o jucărie aparţine unui singur copil, însă două „instanţe" ale aceleiaşi jucării nu pot fi diferenţiate prin denumire, iar acelaşi tip de jucărie poate fi cumpărat mai multor copii, aşa că:
  `Jucărie` ⟶/ `CodFamilie`;
  `Jucărie` ⟶/ `NumeFamilie`;
  `Jucărie` ⟶/ `Copil`;
  `Jucărie` ⟶/ `DataCumpărării`;
- dacă un copil are o singură jucărie de acelaşi tip, se poate scrie:
  (2) (`CodFamilie`, `Copil`, `Jucărie`) ⟶ `DataCumpărării`[4].

Relaţia LOCALITĂŢI_CODURI_VECHI

Continuăm cu relaţia din figura 3.15 (paragraful 3.4), care se referă la vechiul sistem de codificare poştală:

- pentru un oraş sau comună exista un singur cod poştal:
  (1) `CodPoştal` ⟶ `OraşComună`;
  (2) `CodPoştal` ⟶ `Judeţ`;

4. De altminteri, această DF decurge din însăşi calitatea de cheie primară a sursei.

- într-o comună, mai multe sate partajau un singur cod poștal:
  CodPoștal —/→ Sat.

Relația CODURI_NOI_V2

Pentru noul sistem de codificare poștală lucrurile stau diferit. Astfel, condițiile care guvernează relațiile tabelei din figura 3.17 (paragraful 3.4) pot fi sistematizate după cum urmează:

- un cod poștal nu poate fi alocat la două localități, cu atât mai puțin la două comune sau județe diferite:
  (1) CodPoștal ⟶ Localitate;
  (2) CodPoștal ⟶ Comuna;
  (3) CodPoștal ⟶ Județ;
- același cod poate fi asociat imobilelor din două străzi diferite:
  CodPoștal —/→ Strada, CodPoștal —/→ TipNr,
  CodPoștal —/→ NrIniInițial, CodPoștal —/→ LitInițială;
- porțiuni din aceeași stradă pot avea coduri diferite:
  (Localitate, Strada) —/→ CodPoștal;
- intervalele adreselor sunt disjuncte la acordarea codurilor, așa că putem formula și două DF cu sursa compusă:
  (4) (CodPoștal, Strada, NrInițial, LitInițială) ⟶ NrFinal;
  (5) (CodPoștal, Strada, NrInițial, LitInițială) ⟶ LitFinală.

Baza de date VÂNZĂRI

Dat fiind numărul de atribute și specificul aplicației, în acest caz discuțiile sunt ceva mai laborioase:

- fiecare județ are un nume și un indicativ unice:
  (1) Jud ⟶ Regiune;
  (2) Județ ⟶ Regiune;
  (3) Jud ⟶ Județ;
  (4) Județ ⟶ Jud;
- pe baza discuției de la exemplul anterior (CODURI_NOI_V2):
  (5) CodPost ⟶ Localitate;
  (6) CodPost ⟶ Comuna;
  (7) CodPost ⟶ Jud;
  (8) CodPost ⟶ Județ;
- fiecare firmă-client este identificată în mod unic atât prin cod (CodCl):
  (9) CodCl ⟶ DenCl;
  (10) CodCl ⟶ CodFiscal;
  (11) CodCl ⟶ StradaCl;
  (12) CodCl ⟶ NrStradaCl;
  (13) CodCl ⟶ BlocScApCl;
  (14) CodCl ⟶ TelefonCl;
  (15) CodCl ⟶ CodPost;
  (16) CodCl ⟶ Localitate;

(17) CodCl ⟶ Comuna;
(18) CodCl ⟶ Jud;
(19) CodCl ⟶ Judeţ;
(20) CodCl ⟶ Regiune;

- cât şi prin cod fiscal (CodFiscal):
  (21) CodFiscal ⟶ DenCl;
  (22) CodFiscal ⟶ CodCl;
  (23) CodFiscal ⟶ StradaCl;
  (24) CodFiscal ⟶ NrStradaCl;
  (25) CodFiscal ⟶ BlocScApCl;
  (26) CodFiscal ⟶ TelefonCl;
  (27) CodFiscal ⟶ CodPost;
  (28) CodFiscal ⟶ Localitate;
  (29) CodFiscal ⟶ Comuna;
  (30) CodFiscal ⟶ Jud;
  (31) CodFiscal ⟶ Judeţ;
  (32) CodFiscal ⟶ Regiune;
- elementul care identifică un produs este codul acestuia:
  (33) CodPr ⟶ DenPr;
  (34) CodPr ⟶ UM;
  (35) CodPr ⟶ Grupa;
  (36) CodPr ⟶ ProcTVA;
- o factură se întocmeşte pentru un singur client:
  (37) NrFact ⟶ CodCl;
  (38) NrFact ⟶ CodFiscal;
- o factură se întocmeşte o singură dată:
  (39) NrFact ⟶ DataFact;
- în plus:
  (40) NrFact ⟶ Obs;
- o factură conţine mai multe linii: NrFact ⟶/ Linie;
- fiecare linie dintr-o factură se referă la vânzarea unui produs (CodPr), indicând cantitatea şi preţul unitar la care s-a făcut vânzarea:
  (41) (NrFact, Linie) ⟶ Cantitate;
  (42) (NrFact, Linie) ⟶ PreţUnit;
  (43) (NrFact, Linie) ⟶ CodPr;
  (44) (NrFact, Linie) ⟶ DenPr;
  (45) (NrFact, Linie) ⟶ UM;
  (46) (NrFact, Linie) ⟶ ProcTVA;
- într-o factură, un produs apare pe o singură linie:
  (47) (NrFact, CodPr) ⟶ Cantitate;
  (48) (NrFact, CodPr) ⟶ PreţUnit;
  (49) (NrFact, CodPr) ⟶ Linie;
- o încasare are la bază un document primar (*Ordin de plată*, *Chitanţă* etc.):
  (50) CodÎnc ⟶ DataÎnc;
  (51) CodÎnc ⟶ CodDoc;

(52) CodÎnc ⟶ NrDoc;
(53) CodÎnc ⟶ DataDoc;

- documentele de încasare pot prezenta numere similare de la un client la altul; spre exemplu, *Ordinul de plată nr. 101* poate fi emis de mai mulţi clienţi (bineînţeles, este vorba de trei documente diferite care au însă acelaşi număr, poate chiar şi dată):
  (CodDoc, NrDoc) —/→ DataDoc;
  (CodDoc, NrDoc) —/→ NrFact;
  (CodDoc, NrDoc) —/→ CodCl;
- o factură este încasată în mai multe tranşe: NrFact —/→ CodÎnc;
- o încasare poate achita mai multe facturi: CodÎnc —/→ NrFact;
- într-o încasare, o tranşă se referă la o singură factură:
  (54) (CodÎnc, NrFact) ⟶ Tranşă,
- nu însă şi (CodÎnc, Tranşă) ⟶ NrFact, deoarece într-o aceeaşi încasare pot exista două tranşe egale (de exemplu: 5 000 000 lei) referitoare la două facturi diferite.

4.1.3. *Proprietăţi ale dependenţelor funcţionale, închidere şi acoperire minimală*

Dependenţele funcţionale prezintă câteva proprietăţi pe care le discutăm, în cea mai mare parte, pe calapodul relaţiei VÂNZĂRI.

1. *Reflexivitate*[5] sau *incluziune*[6]: dacă $Y \subseteq X$, atunci X ⟶ Y, adică un ansamblu oarecare de atribute determină pe oricare dintre atributele sale. Exemplu: Y = NrFact, X = (CodÎnc, NrFact). Se poate scrie: (CodÎnc, NrFact) ⟶ NrFact.

2. *Extensie* (*creştere* sau *multiplicare*): dacă X ⟶ Y, atunci XZ ⟶ YZ. Cu alte cuvinte, dacă X determină pe Y, cele două ansambluri de atribute pot fi „îmbogăţite" printr-un al treilea. Exemplu: NrFact ⟶ CodCl, prin urmare (NrFact, CodÎnc) ⟶ (CodCl, CodÎnc).

3. *Tranzitivitate*: dacă X ⟶ Y şi Y ⟶ Z, atunci X ⟶ Z. Exemplu: deoarece NrFact ⟶ CodCl şi CodCl ⟶ DenCl, atunci NrFact ⟶ DenCl. Practic, dacă aplicăm această proprietate, obţinem următoarele DF:

(55) NrFact ⟶ DenCl;
(56) NrFact ⟶ CodFiscal;
(57) NrFact ⟶ StradaCl;

5. Denumirea de *reflexivitate* apare în [Date 04], p. 338, [Elmasri & Navathe 00], p. 479, [Garcia-Molina ş.a. 02], p. 99 etc.
6. Termenul *incluziune* este folosit în [O'Neil & O'Neil 01], p. 362.

(58) NrFact ⟶ NrStradaCl;

(59) NrFact ⟶ BlocScApCl;

(60) NrFact ⟶ TelefonCl;

(61) NrFact ⟶ CodPost;

(62) NrFact ⟶ Localitate;

(63) NrFact ⟶ Comuna;

(64) NrFact ⟶ Jud;

(65) NrFact ⟶ Judeţ;

(66) NrFact ⟶ Regiune.

Aceste trei proprietăţi ale dependenţelor funcţionale sunt denumite *axiomele lui Armstrong*. Alte reguli se referă la:

4. *Autodeterminare*: X ⟶ X.

5. *Uniune*: dacă X ⟶ Y şi X ⟶ Z, atunci X ⟶ YZ. Exemplu: întrucât NrFact ⟶ CodCl şi NrFact ⟶ DataFact, rezultă NrFact ⟶ (CodCl, DataFact).

6. *Pseudo-tranzitivitate*: dacă X ⟶ Y şi WY ⟶ Z, atunci WX ⟶ Z. Să discutăm cazul relaţiei CĂRŢI_AUT {Cota, ISBN, Autor, NrPagini}, unde NrPagini indică numărul de pagini din cartea curentă scrise de autorul curent (din fiecare tuplu al relaţiei). Întrucât Cota ⟶ ISBN şi (Autor, ISBN) ⟶ NrPagini, este valabilă şi dependenţa (Autor, Cota) ⟶ NrPagini.

7. *Descompunere*: dacă X ⟶ YZ, atunci X ⟶ Y şi X ⟶ Z. Este opusă uniunii.

Hugh Darwen a formulat *Teorema unificării generale* (*General Unification Theorem*), prin care o parte dintre regulile de mai sus devin cazuri particulare ale următorului enunţ:

8. Dacă X ⟶ Y şi Z ⟶ W, atunci X ∪ (Z − Y) ⟶ YW[7].

Chiar pentru cazuri simple, ansamblul tuturor DF într-o bază de date poate să devină extrem de stufos. De aceea, în procesul normalizării trebuie avut în vedere un echilibru între:

- identificarea tuturor DF ce reflectă restricţii, reguli ale fenomenului/procesului modelat, şi

7. Hugh Darwen, *The Role of Functional Dependence in Query Decomposition*, în Date, C.J., Darwen, H., *Relational Database Writings 1989-1991*, Addison-Wesley, Reading, Mass., 1992.

- includerea în setul DF numai a celor relevante, adică a unui ansamblu minimal din care, prin reguli de genul celor de mai sus, se pot obţine toate celelalte DF.

În teorie, dat fiind un set de DF notat S, ansamblul tuturor DF determinate pe baza acestuia se numeşte *închidere* a lui S şi se notează S^+. Multe lucrări consacrate bazelor de date prezintă algoritmi pentru calculul închiderilor, atât pentru toată relaţia, cât şi pentru un grup de atribute, dat fiind un set de DF de plecare. Utilitatea închiderilor în practica proiectării bazelor de date rămâne încă de demonstrat.

Mult mai importantă este determinarea *acoperirii minimale*, adică a setului minim de DF prin care pot fi determinate toate celelalte dependenţe din relaţie. După Chris Date, un set S de dependeţe funcţionale este ireductibil dacă şi numai dacă satisface următoarele trei proprietăţi[8]:

1. Partea dreaptă (destinaţia) a fiecărei dependenţe din S implică un singur atribut.
2. Sursa fiecărei DF este ireductibilă, adică nici un atribut nu poate fi eliminat fără distrugerea dependenţei; se spune că DF este ireductibilă la stânga.
3. Nici o DF din S nu poate fi eliminată fără afectarea închiderii S^+, altfel spus, fără pierderea unei serii de informaţii.

Din păcate, deşi elegant postulată, acoperirea minimală ridică probleme majore chiar şi la un număr mediu de atribute, ca să nu vorbim cât de alunecoasă este proprietatea a treia formulată de Date. În plus, de multe ori, pentru o bază de date există mai multe acoperiri minimale[9]. Lucrurile pot fi însă simplificate parţial eliminând din discuţie dependenţele simetrice şi redundante.

4.1.4. *Dependenţe funcţionale simetrice şi redundante*

Deseori, o bază de date sau relaţie prezintă mai multe chei candidate. Alegerea dintre acestea a cheii primare are în vedere, pe lângă unicitate, compoziţia minimală şi valorile nenule ale oricărui atribut component, precum şi elemente ce ţin de facilitatea utilizării: uşurinţa reţinerii (de către utilizator), lungime cât mai mică, constanţă în timp etc.

În procesul normalizării, problemele ridicate de aceste dependenţe reciproce dintre cheile candidate trebuie tratate cu prudenţă. De cele mai multe ori, este recomandat ca dintre dependenţele unei relaţii să fie eliminate toate cele în care sursa este o cheie candidată şi păstrarea numai a celor în care determinantul este atributul cheie primară.

Revenim la cele 54 de dependenţe din relaţia universală VÂNZĂRI tratate în finalul paragrafului 4.1.2. Este evident că dependenţele (3), `Jud` ⟶ `Judeţ`, şi (4), `Judeţ` ⟶ `Jud`, sunt simetrice. Deşi corecte, dependenţele (1), `Jud` ⟶ `Regiune`, şi (2), `Judeţ` ⟶ `Regiune`, „se calcă pe picior". Pentru comoditate, putem alege atributul `Jud` ca identificator „de bază" al unui judeţ, eliminând (pe nedrept), pe lângă (4), toate DF în care sursa simplă este atributul `Judeţ`, iar destinaţiile identice

8. [Date 00], p. 338. Vezi şi [Garcia-Molina 00], p. 482.
9. [Garcia-Molina00], p. 482.

cu cele ale dependenţelor în care sursă este `Jud`. Ce-i drept, efortul nu este chiar supraomenesc, deoarece este o singură dependenţă de acest tip, (2).

La fel se pune problema şi pentru perechile de atribute `CodCl` şi `DenCl` pe de o parte şi `CodCl` şi `CodFiscal` pe de altă parte. Toate identifică un acelaşi tip de entitate – o firmă client. Învestim `CodCl` drept „reprezentant unic" al unui client. `DenCl` nefiind luat prea mult în seamă în setul de DF, avem de eliminat doar DF în care `CodFiscal` este sursă simplă: (21)-(32). În privinţa perechii `CodPr` şi `DenPr` nu avem a ne epuiza, deoarece, aproape intenţionat, au fost omise din setul DF cele în care `DenPr` apare ca sursă.

Atenţie! Nu facem acelaşi lucru în relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 cu atributele `Cota` şi `ISBN`, deoarece primul se referă la un exemplar „fizic" al unei cărţi din bibliotecă, în timp ce al doilea se referă la un titlu (carte publicată la o editură).

Tipul acesta de redundanţă e uşor de identificat şi eliminat, chiar dacă nu se leagă vizibil de nici o teoremă sau lemă ce constituie fundamentul normalizării. Setul DF conţine însă şi un alt tip de redundanţă/circularitate. Astfel, deoarece fiecare linie dintr-o factură se referă la un produs vândut la un preţ unitar şi într-o anumită cantitate, sursele DF în care destinaţiile sunt `Cantitate`, `PreţUnit` (plus `DenPr`, `UM` şi `ProcTVA`) pot fi atât (`NrFact`, `Linie`), cât şi (`NrFact`, `CodPr`). Procedând analog situaţiei de mai sus (dar cu mai multă strângere de inimă), decidem ca fiecare linie/produs dintr-o factură să fie identificat de sursa (`NrFact`, `Linie`). Aşa că eliminăm dependenţele în care sursa este „concurentul", adică (`NrFact`, `CodPr`), şi destinaţiile sunt identice: (47), (48) şi (49).

4.2. Reprezentări ale dependenţelor funcţionale

Elaborarea structurii bazelor de date prin tehnica normalizării necesită o gestiune riguroasă a tuturor dependenţelor funcţionale. În practică, parcurgerea tuturor relaţiilor dintre atributele unei baze de date se poate solda cu identificarea a zeci sau chiar sute de dependenţe. Conectarea lor, determinarea redundanţelor, incluziunilor, tranzitivităţilor ridică astfel uneori probleme apăsătoare. Drept pentru care au fost elaborate diverse instrumente menite a simplifica vizualizarea şi lucrul cu DF. În acest paragraf vom prezenta trei asemenea instrumente: diagrame, grafuri şi matrice ale dependenţelor funcţionale.

4.2.1. *Diagrame ale DF*

Diagramele DF au o utilizare limitată[10], deoarece la un număr mare de atribute reprezentarea dependenţelor este foarte anevoioasă. O dependenţă dintre două atribute

10. Lucrarea din care am preluat diagramele DF este [Pratt & Adamski 91], p. 261. La noi, lucrarea cea mai cunoscută în care sunt folosite este [Oprea 99].

este indicată printr-o linie (sau un număr de segmente) ce porneşte din sursă şi se termină printr-o săgeată „înfiptă" în destinaţie (determinant). Când sursele sau destinaţiile sunt compuse, atunci, mai întâi, se unesc atributele din grupul determinant/destinaţie, iar terminaţiile dependenţelor vizează segmentele de grup.

O porţiune din relaţia VÂNZĂRI

Dacă luăm în considerare DF de la (33) la (36) şi de la (41) la (49), discutate în paragraful 4.1.2, diagrama acestora ar arăta ca în figura 4.2.

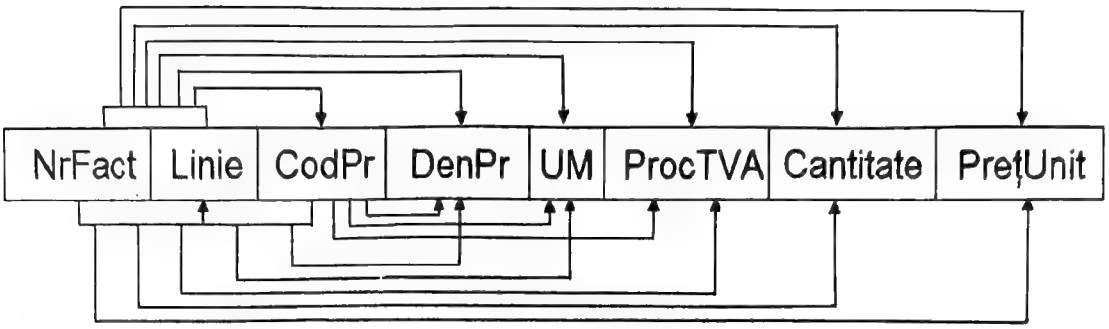


Figura 4.2. Diagrama DF – o parte din atributele relaţiei VÂNZĂRI

După cum se observă, există un grad apăsător de redundanţă, aşa încât ne vom conforma recomandărilor din paragraful 4.1.4 şi vom elimina DF simetrice/redundante, obţinând diagrama din figura 4.3.

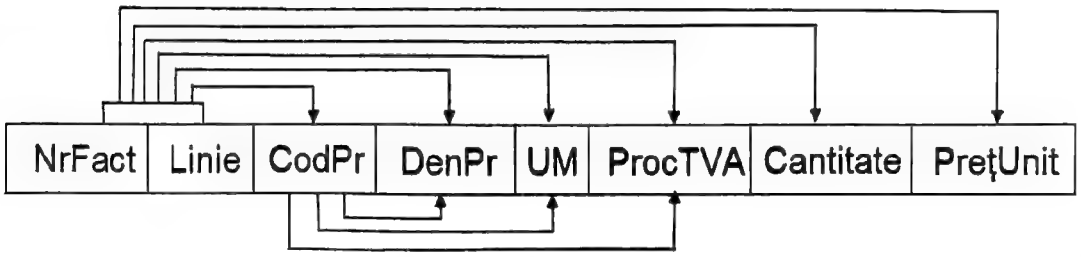


Figura 4.3. Diagrama DF din figura 4.2, însă fără dependenţele redundante

Relaţia LINII_ARTICOLE_CONTABILE

Pentru relaţia din figura 4.1, dacă se foloseşte primul sistem de contare, constant de la lună la lună, în care dependenţa funcţională este (1), diagrama are forma din figura 4.4, iar pentru al doilea tip, cel în care se manifestă dependenţele (2), (3) şi (4), diagrama este cea din figura 4.5.

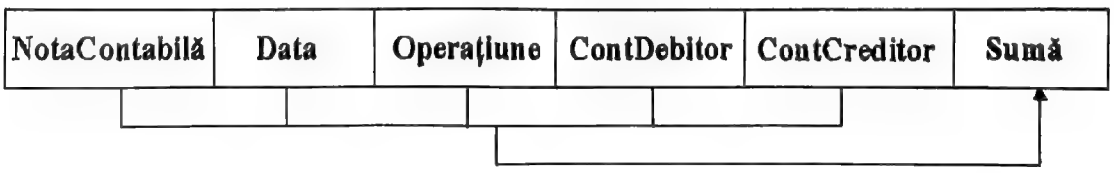


Figura 4.4. Diagrama DF – exemplul 1: primul tip de contare

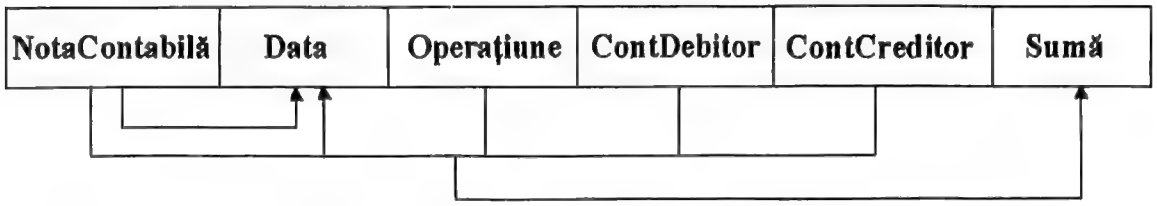


Figura 4.5. Diagrama DF – exemplul 2: al doilea tip de contare

Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3

Diagrama dependenţelor acestei relaţii (figura 3.7), dependenţe amintite în paragraful 4.1.2, este prezentată în figura 4.6.

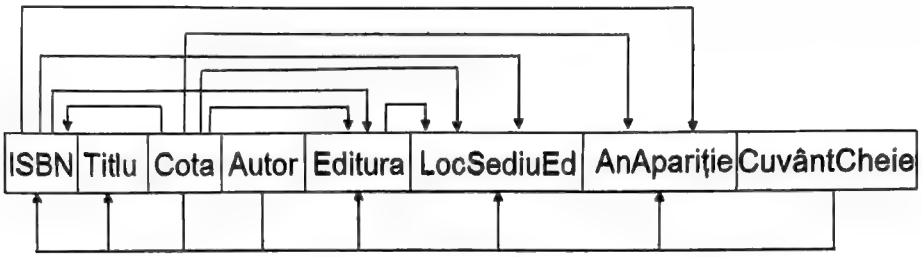


Figura 4.6. Diagrama DF – exemplul 3

Dacă pentru exemplele de mai sus diagramele pot fi trasate cu oarecare uşurinţă, pentru exemplul 7 din paragraful 4.1.2, cel dedicat bazei de date VÂNZĂRI, în care apar mult mai multe atribute şi dependenţe, diagrama este practic imposibil de reprezentat pe o coală A4. În asemenea situaţii, care formează „grosul" cazurilor practice, mult mai la îndemână sunt graful şi matricea DF.

4.2.2. *Graful dependenţelor funcţionale*

Într-un graf al dependenţelor funcţionale, acestea sunt reprezentate prin săgeţi ce leagă sursa şi destinaţia. La dependenţele funcţionale simple săgeata uneşte cele două atribute. Când sursa este compusă, se foloseşte un conector care leagă, în primă instanţă, atributele determinant, iar săgeata uneşte acest conector cu atributul destinaţie. Astfel, diagrama din figura 4.3 îmbracă forma grafului din figura 4.7.

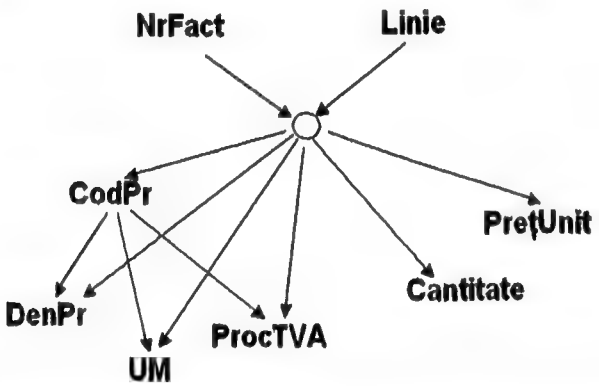


Figura 4.7. Reprezentarea sub formă de graf a diagramei din figura 4.3

Cât priveşte dependenţele exemplului 3 (paragraful 4.1.2), corespunzător diagramei din figura 4.6 este graful din figura 4.8. În ciuda densităţii de săgeţi, se remarcă un plus de claritate şi folosirea mai judicioasă a spaţiului.

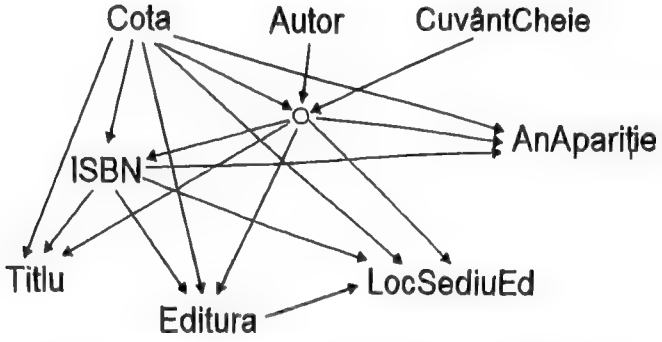


Figura 4.8. Reprezentarea sub formă de graf a diagramei din figura 4.6.

Spre deosebire de diagramă, cu graful DF se poate reprezenta voluminosul ansamblu al DF pentru baza de date VÂNZĂRI - exemplul 7 (paragraful 4.1.2). Pentru simplificare, vom reprezenta numai DF (1), (3), (5)-(7), (9)-(15), (33)-(46), (50)-(54), adică, după cum vom vedea în pagraful 5.1, acoperirea minimală - vezi figura 4.9[11].

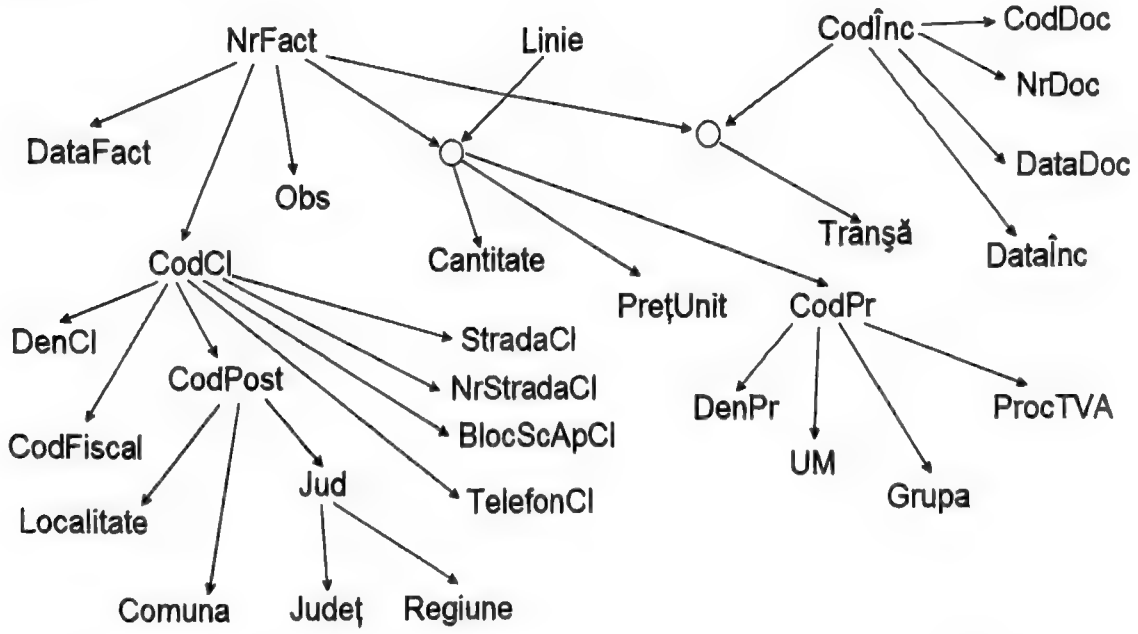


Figura 4.9. Graful DF corespunzător acoperirii minimale a BD VÂNZĂRI

11. De fapt, eliminăm atât dependenţele simetrice şi redundante, cât şi pe cele tranzitive, luând-o astfel înainte cu „materia", deoarece acesta este subiectul capitolului următor.

4.2.3. *Matricea dependenţelor funcţionale*

Construirea matricei dependenţelor funcţionale demarează prin crearea unui tabel în care fiecărui atribut i se va asocia o linie şi o coloană. După includerea tuturor atributelor, se adaugă toate grupurile de atribute care constituie surse de dependenţe funcţionale. Apoi se parcurg, pe rând, coloanele, în fiecare celulă scriindu-se 1 atunci când atributul de pe coloană este sursă a unei DF în care destinaţia este atributul/ grupul de pe linie.

Astfel, corespunzătoare diagramei din figura 4.3 şi grafului din figura 4.7, matricea DF s-ar putea prezenta ca în figura 4.10.

		Surse de dependenţe funcţionale								
		1	2	3	4	5	6	7	8	9
1	NrFact									
2	Linie									
3	CodPr									1
4	DenPr			1						1
5	UM			1						1
6	ProcTVA			1						1
7	Cantitate									1
8	PreţUnit									1
9	(NrFact, Linie)									

Figura 4.10. Matricea corespunzătoare diagramei din fig. 4.2. şi grafului din fig. 4.10.

Pentru simplificarea figurii, de obicei, se recurge la eliminarea coloanelor care nu sunt surse de dependenţe funcţionale, ca în figura 4.11.

		Surse de DF	
		3	9
1	NrFact		
2	Linie		
3	CodPr		1
4	DenPr	1	1
5	UM	1	1
6	ProcTVA	1	1
7	Cantitate		1
8	PreţUnit		1
9	(NrFact, Linie)		

Figura 4.11. Matricea anterioară simplificată

Păstrând paralelismul, figura 4.12 prezintă matricea DF corespunzătoare diagramei din figura 4.6 şi grafului din figura 4.8.

Firește, cea mai complicată matrice este cea a bazei de date VÂNZĂRI (exemplul 7 (paragraful 4.1.2), figura 4.13 fiind transpunerea grafului din figura 4.9.

		Surse de DF			
		1	2	6	9
1	Cota				
2	ISBN	1			1
3	Autor				
4	CuvântCheie				
5	Titlu	1	1		1
6	Editura	1	1		1
7	LocSediuEd	1	1	1	1
8	AnApariţie	1	1		1
9	(Cota, Autor, CuvântCheie)				

Figura 4.12. Matricea corespunzătoare diagramei din figura 4.6 şi grafului din figura 4.8

		Surse DF							
		1	4	7	14	19	25	31	32
1	Jud								
2	Judeţ	1							
3	Regiune	1							
4	CodPost								
5	Localitate		1						
6	Comună		1						
7	CodCl								
8	DenCl			1					
9	CodFiscal			1					
10	StradaCl			1					
11	NrStradaCl			1					
12	BlocScApCl			1					
13	TelefonCl			1					
14	CodPr							1	
15	DenPr				1				
16	UM				1				
17	Grupa				1				
18	ProcTVA				1				
19	NrFact								
20	DataFact					1			
21	Obs					1			
22	Linie								
23	Cantitate							1	
24	PreţUnit							1	
25	CodÎnc								
26	DataÎnc						1		
27	CodDoc						1		
28	NrDoc						1		
29	DataDoc						1		
30	Tranşă								1
31	(NrFact, Linie)								
32	(CodÎnc, NrFact)								

Figura 4.13. Matricea simplificată a DF ale bazei de date VÂNZĂRI

4.3. Dependenţe funcţionale totale

Noţiunea de *dependenţă funcţională totală* a fost introdusă de Melkanoff. O dependenţă funcţională `Data1` ⟶ `Data2` este *totală* dacă nu există nici un atribut/combinaţie de atribute `Dată3` ca subansamblu al `Data1` care să verifice dependenţa funcţională `Dată3` ⟶ `Dată2`.

Formal, dacă $X = \{A_i, A_j, \ldots, A_k\}$ reprezintă un grup de atribute ale relaţiei R, iar A_m un atribut al aceleiaşi relaţii, unde $A_m \notin X$, se spune că există o dependenţă funcţională totală între X şi A_m dacă:

1. $X \longrightarrow A_m$ şi
2. nu există nici un subansamblu S de atribute, $S \subset X$, pentru care $S \longrightarrow A_m$.

Se notează: $X \xrightarrow{total} A_m$

Dependenţa funcţională totală mai este denumită şi *elementară, plină, deplină* sau *completă*. Implicit, toate dependenţele în care sursa este simplă (alcătuită dintr-un singur atribut) sunt dependenţe funcţionale complete. Problema prezintă importanţă în cazul dependenţei funcţionale cu sursa compusă. Astfel, în BD VÂNZĂRI, dependenţele:

(`NrFact, Linie`) ⟶ `CodPr`

(`NrFact, Linie`) ⟶ `Cantitate`

(`NrFact, Linie`) ⟶ `PretUnit`

sunt toate totale, deoarece: `NrFact` ⟶/ `CodPr`, `NrFact` ⟶/ `Cantitate`, `NrFact` ⟶/ `PretUnit`, `Linie` ⟶/ `CodPr`, `Linie` ⟶/ `Cantitate`, `Linie` ⟶/ `PretUnit`.

Dacă se utilizează cel de-al doilea sistem de contare, caz în care sunt valabile DF (2)-(4), atunci pentru LINII_ARTICOLE_CONTABILE din figura 4.1 dependenţa (4) este una parţială datorită (2).

În relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 din figura 3.7, primele cinci dependenţe funcţionale:

(1) (`Cota, Autor, CuvântCheie`) ⟶ `ISBN`

(2) (`Cota, Autor, CuvântCheie`) ⟶ `Titlu`

(3) (`Cota, Autor, CuvântCheie`) ⟶ `Editura`

(4) (`Cota, Autor, CuvântCheie`) ⟶ `LocSediuEd`

(5) (`Cota, Autor, CuvântCheie`) ⟶ `AnApariţie`

sunt parţiale, deoarece:

(6) `Cota` ⟶ `ISBN`

(7) `Cota` ⟶ `Titlu`

(8) `Cota` ⟶ `Editura`

(9) `Cota` ⟶ `LocSediuEd`

(10) `Cota` ⟶ `AnApariţie`

Cu alte cuvinte, un atribut component al determinatului dependenţelor (1)-(5) este, el singur, sursă într-o serie de DF în care destinaţiile sunt identice.

Ansamblul DF ce desemnează acoperirea minimală a relaţiei universale VÂNZĂRI, ansamblu reprezentat în paragraful anterior sub formă de graf (figura 4.9) şi matrice (figura 4.13), nu are dependenţele parţiale (ca din întâmplare).

4.4. Trecerea relaţiilor în a doua formă normală (2NF)

Începând cu a doua formă normală, relaţiile pot fi decupate în subrelaţii, în scopul diminuării problemelor legate de stocare şi actualizare. În general, atunci când într-o relaţie R există o DF de tip `X` ⟶ `Y` şi `X` nu este cheie candidată, atunci R conţine o anumită doză de redundanţă[12].

Heath a demonstrat[13] că orice relaţie alcătuită din trei atribute, notată R(`X`, `Y`, `Z`), în care există dependenţa funcţională `X` ⟶ `Y` poate fi descompusă în două relaţii R1(`X`, `Y`) şi R2(`X`, `Z`); R1 şi R2 reprezintă proiecţiile relaţiilor R pe atributele `X` şi `Y`, respectiv `X` şi `Z`. Esenţial este faptul ca descompunerea să se facă fără pierdere de informaţii, adică R să poată fi „recompusă" prin joncţiunea tabelelor R1 şi R2. În alţi termeni, la spargerea oricărei relaţii trebuie ca toate dependenţele din acoperirea minimală să se regăsească în structura noilor relaţii[14]. Ca o consecinţă indirectă, se poate astfel rezolva şi problema cheii primare a relaţiei universale, în cazul încălcării restricţiei de entitate (un atribut din cheia primară prezintă valori nule).

O relaţie se află în 2NF dacă:

1. Se află în 1NF.
2. Toate DF ce leagă cheia primară la celelalte atribute sunt DF elementare (totale).

O bază de date este în 2NF când toate relaţiile care o alcătuiesc sunt în 2NF. Problema trecerii unei relaţii din prima în a doua formă normală se pune numai când cheia primară a relaţiei este compusă din mai multe atribute. Într-o formulare mai lejeră, se poate spune că *o relaţie în 2NF nu conţine DF parţiale între atributele cheie şi celelalte atribute*.

În general, trecerea de la 1NF la 2NF se poate realiza după următoarea succesiune de 6 paşi:

1. Se identifică posibila cheie primară a relaţiei universale, care conferă unicitate oricărui tuplu din relaţie, chiar în condiţiile violării restricţiei de entitate.

12. [Date 00], p. 333.
13. Heath, I.J., *Unacceptable File Operations in a Relational Database*, Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, noiembrie 1971.
14. Vezi şi [O'Neil & O'Neil 01], p. 387.

2. Se inventariază toate dependenţele dintre atributele relaţiei (închiderea DF), cu excepţia celor în care destinaţia este un atribut component al cheii primare.
3. Se trec în revistă toate dependenţele care au ca sursă (determinant) un atribut sau subansamblu de atribute din cheia primară.
4. Pentru fiecare atribut (sau subansamblu) al cheii de la pasul 3 se creează o relaţie care va avea drept identificator primar atributul (subansamblul) respectiv, iar celelalte atribute vor fi cele care apar ca destinaţii în dependenţele de la pasul 3.
5. Din relaţia iniţială sunt eliminate toate atributele destinaţie (non-cheie) din relaţiile „proaspăt" obţinute, păstrându-se numai atributele cheie ale noilor relaţii (se asigură, astfel, cheile străine care vor face legăturile către noile relaţii).
6. În „rămăşiţele" relaţiei universale (iniţiale), se verifică dacă cheia primară iniţială respectă restricţia de entitate, iar dacă nu, se mai elimină dintre atribute, la limită, chiar toată relaţia-„rămăşiţe".

Important! Etapele sugerate sunt aplicabile mai ales în situaţiile în care cheia primară a relaţiei universale încalcă restricţia de entitate, adică atunci când combinaţia ar conferi unicitate, însă unul sau mai multe dintre atributele cheie ar risca valori nule (vezi discuţia din paragraful 3.4). Este vorba de violarea ruşinoasă a „poruncilor" relaţionale, deoarece, după cum am promis, începând cu 2NF, orice relaţie obţinută va respecta scrupulos (şi) restricţia de entitate.

La limită, prin descompuneri, din relaţia universală (iniţială) este posibil să nu rămână decât atributele cheie. Acum, însă, orice pericol privind nulitatea vreunui atribut cheie va atrage după sine eliminarea atributului respectiv sau, la limită, a relaţiei respective, după cum vom vedea chiar în acest paragraf, pentru relaţia VÂNZĂRI.

Relaţia LINII_ARTICOLE_CONTABILE

Pentru relaţia din figura 4.1, dacă se utilizează primul sistem de contare, în care singura dependenţă dintre cele discutate în paragraful 4.1.2 este (1), atunci nu se mai pune problema trecerii în 2NF. În al doilea caz, în care sunt valabile DF (2), (3) şi (4), ultima este parţială. Urmăm cei 6 paşi:

1. Cheia primară este combinaţia: (`NotaContabilă`, `Operaţiune`, `ContDebitor`, `ContCreditor`).
2. Inventarierea tuturor DF: (2), (3), (4).
3. Singura dependenţă în care determinantul unui atribut din cheia primară este (2): `NotaContabilă` ⟶ `Data`.
4. Pentru atributul `NotaContabilă` se constituie o relaţie care îl va avea identificator primar, iar cel de-al doilea atribut va fi `Data`. Din tabela iniţială se elimină atributul necheie preluat în noua relaţie.
5. Din relaţia iniţială rămâne: R' (`NotaContabilă`, `Operaţiune`, `ContDebitor`, `ContCreditor`, `Suma`).
6. În R', cheia primară nu periclitează restricţia entităţii, deci R' îşi poate păstra structura.

Prin urmare, în 2NF, relaţia LINII_ARTICOLE_CONTABILE se descompune în două relaţii, NOTE_CTB şi ARTICOLE_CTB, prezentate în figura 4.14.

NOTE_CTB {NotaContabilă, Data}

ARTICOLE_CTB {NotaContabilă, Operaţiune, ContDebitor, ContCreditor, Suma}

NOTE_CTB

Nota_Contabilă	Data
150	30.11.2000
151	01.12.2000

ARTICOLE_CTB

Nota_Contabilă	Operatiune	Cont_Debitor	Cont_Creditor	Suma
150	1	300	401	1 000 000
150	1	4426	401	180 000
150	2	401	5311	700 000
150	2	401	5121	480 000
151	1	5121	700	525 000

Figura 4.14. Relaţia LINII_ARTICOLE_CONTABILE adusă în 2NF

La drept vorbind, nu putem spune că economia de spaţiu şi diminuarea anomaliilor la actualizări sunt spectaculoase din cale-afară prin aducerea relaţiei în 2NF. Din raţiuni de simplitate, exemplul este însă util.

Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3

Relaţia ilustrată în figura 3.7 prezintă, dintre cele discutate în paragraful 3.3.2, dependenţe parţiale şi, astfel, constituie un bun „obiect" de normalizare în 2NF. De altminteri, relaţia abundă în redundanţe şi anomalii de genul celor discutate în paragraful 2.1:

- înregistrarea recepţiei unui titlu, introducerea datelor despre o carte (titlu) sunt imposibile până în momentul în care măcar o carte primeşte cotă;
- dacă un titlu a fost introdus iniţial eronat, corecţia trebuie operată pe destule linii ale tabelei;
- la fiecare achiziţie ulterioară a aceleiaşi cărţi (ISBN) trebuie introduse din nou ISBN-ul, titlul, editura etc.

Parcurgem cei 6 paşi, după cum urmează:

1. Cheia primară a relaţie iniţiale este: (Cota, Autor, CuvântCheie).
2. Ansamblul DF: (1)-(15) – vezi paragraful 4.1.2.
3. În dependenţele (6)-(10), atributul Cota, component al cheii relaţiei, este sursă simplă.
4. Se constituie o relaţie în care Cota este cheie primară, iar celelalte atribute sunt destinaţiile dependenţelor (6), (7), (8), (9) şi (10).

5. Din relaţia iniţială se elimină toate atributele destinaţie preluate în proaspăta tabelă, rămânând doar atributele cheie: {Cota, Autor, CuvântCheie}.
6. Nici unul dintre cele trei atribute de la punctul 5 nu periclitează restricţia de entitate, aşa încât relaţia – „rămăşiţă" se păstrează în această formă.

Prin urmare, în a doua formă normalizată, relaţia iniţială (universală) s-ar putea descompune în:

CĂRŢI {Cota, ISBN, Titlu, Editura, LocSediuEd, AnApariţie}

şi

TITLURI_AUTORI_CUVINTECHEIE {Cota, Autor, CuvântCheie}

Conţinutul acestor două noi relaţii este prezentat în figura 4.15.

CĂRŢI

Cotă	ISBN	Titlu	Editura	LocSediuEd	AnApariţie
III-13421	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Polirom	Iaşi	2002
III-13422	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Polirom	Iaşi	2002
III-13423	973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Polirom	Iaşi	2002
III-10678	973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Polirom	Iaşi	2001
III-10679	973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Polirom	Iaşi	2001

TITLURI_AUTORI_CUVINTECHEIE (fragment)

Cota	Autor	CuvântCheie
III-13421	Marin Fotache	baze de date
III-13421	Marin Fotache	SQL
III-13421	Marin Fotache	proceduri stocate
III-13421	Marin Fotache	FoxPro
III-13421	Marin Fotache	formulare
III-13421	Marin Fotache	orientare pe obiect
III-13421	Marin Fotache	client-server
III-13421	Marin Fotache	Web
	...	

Figura 4.15. Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 în 2NF

Chiar dacă o parte dintre redundanţe au fost eliminate, mai rămân destule, astfel încât structura de mai sus va constitui subiect atât pentru a treia formă normalizată (3NF), cât şi pentru cea de-a patra (4NF).

Relaţia DOTARE_JUCĂRII_1

Nici tabela din figura 3.14 nu se află în 2NF, deoarece cheia primară este compusă – (CodFamilie, Copil, Jucărie), aşa că:

(1) (CodFamilie, Copil, Jucărie) ⟶ NumeFamilie

(2) (CodFamilie, Copil, Jucărie) ⟶ DataCumpărării

iar, după cum am discutat în paragraful 4.1.2,

(3) `CodFamilie` ⟶ `NumeFamilie`

Prin urmare, DF (1) este parţială, iar relaţia se rupe în două tabele, după cum sugerează figura 4.16.

FAMILII

CodFamilie	NumeFamilie
1111	Popescu Ioan

JUCĂRII

CodFamilie	Copil	Jucărie	DataCumpărării
1111	Marc	Trenuleţ electric	14-05-1998
1111	Loredana	Păpuşă Barbie	15-09-1998
1111	Elvis	Puzzle 200 piese	23-12-1998
1111	Loredana	Puzzle 250 piese	14-05-1999
1111	Marc	Puzzle 200 piese	24-12-1999
1111	Loredana	Căsuţa Lego	08-03-2000

Figura 4.16. Relaţia DOTARE_JUCĂRII_1 adusă în 2NF

Relaţia LOCALITĂŢI_CODURI_VECHI

Pentru vechiul sistem de codificare poştală, relaţia LOCALITĂŢI_CODURI_VECHI din figura 3.15, cu cele câteva dependenţe discutate în acelaşi paragraf 4.1.2, nu are cheie primară, deoarece singura combinaţie plauzibilă – (`CodPoştal`, `Sat`) – încalcă restricţia de entitate, întrucât pentru liniile care se referă la oraşe valoarea atributului `Sat` este NULL. Cu toate acestea:

(1) (`CodPoştal`, `Sat`) ⟶ `OraşComună`

(2) (`CodPoştal`, `Sat`) ⟶ `Judeţ`

Şi întrucât:

(3) `CodPoştal` ⟶ `OraşComună`

(4) `CodPoştal` ⟶ `Judeţ`

putem spune că (1) şi (2) sunt DF parţiale.

Ţinând seama de acest lucru, relaţia este ruptă astfel:

ORAŞE_COMUNE {`CodPostal`, `OraşComună`, `Judeţ`}

şi

SATE {`CodPostal`, `Sat`}

al căror conţinut este prezentat în figura 4.17.

Pentru a verifica justeţea ruperii relaţiei iniţiale în cele două noi tabele, ar trebui ca prin joncţiunea ORAŞE_COMUNE cu SATE să obţinem LOCALITĂŢI_CODURI_VECHI. Şi, într-adevăr, prin fraza:

```
SELECT oc.codpostal, orascomuna, sat, judet
FROM orase_comune oc LEFT OUTER JOIN sate s
ON oc.codpostal=s.codpostal
```

ORAŞE_COMUNE

CodPoştal	OraşComună	Judeţ
5319	Vînători	Vrancea
5613	Roznov	Neamţ
5300	Focşani	Vrancea

SATE

CodPoşta	Sat
5319	Vînători
5319	Jorăşti
5319	Mirceştii-Vechi
5613	Roznov
5613	Slobozia

Figura 4.17. Relaţia LOCALITĂŢI_CODURI_VECHI adusă în 2NF cu rezolvarea problemei cheii primare

rezultatul are acelaşi conţinut informaţional ca şi relaţia iniţială. Atenţie însă, folosim *joncţiunea externă*, şi nu pe cea naturală (sau echijoncţiunea)!

Avantajele descompunerii în cele două relaţii sunt vizibile. Mai întâi, ambele sunt în 1NF şi au cheie primară fără nici un risc în privinţa valorilor nule. În al doilea rând, se elimină redundanţele. În LOCALITĂŢI_CODURI_VECHI, dacă o comună are şase sate, de şase ori apare şi codul poştal al comunei, şi denumirea comunei, şi a judeţului. Acum însă, denumirea comunei şi cea a judeţului apar o singură dată în relaţia ORAŞE_COMUNE. De asemenea, pentru fiecare oraş apare o singură înregistrare în ORAŞE_COMUNE şi, fireşte, nici una în SATE. La aşa normalizare reuşită, păcat că vechea codificare poştală nu mai este în vigoare!

Relaţia CODURI_ORAŞE

Pentru noul sistem de codificare poştală - relaţia CODURI_ORAŞE (figura 3.19, paragraful 3.4) nu este în 2NF, deoarece un cod poştal nu poate fi alocat la două oraşe (localităţi):

(1) `CodPoştal` ⟶ `Localitate`

(2) `CodPoştal` ⟶ `Judeţ`

şi, pe de altă parte,

(3) (`CodPoştal, Strada, NrInitial, LitInitială`) ⟶ `Localitate`

(4) (`CodPoştal, Strada, NrInitial, LitInitială`) ⟶ `Judeţ`

aşa că putem rupe relaţia în:

CODURI_OR {`CodPostal, Localitate, Judeţ`}

şi

CODURI_ADRESE {`CodPostal, Strada, NrInitial, LitInitială, TipNr, NrFinal, LitFinală`}

Relaţia CODURI_NOI_V3

Lucrurile se prezintă foarte aproape de relaţia CODURI_ORAŞE discutată mai sus, aşa că putem scrie:

(1) `CodPoştal` ⟶ `Localitate`

(2) `CodPoştal` ⟶ `Comuna`

(3) CodPoştal ⟶ Judeţ

şi, pe de altă parte,

(6) (CodPoştal, Strada, NrIniţial, LitIniţială) ⟶ Localitate

(7) (CodPoştal, Strada, NrIniţial, LitIniţială) ⟶ Comuna

(8) (CodPoştal, Strada, NrIniţial, LitIniţială) ⟶ Judeţ.

Rezultă că putem rupe relaţia în:

CODURI_LOCALITĂŢI {CodPostal, Localitate, Comună, Judeţ}
şi
CODURI_ADRESE {CodPostal, Strada, NrInitial, LitInitială, TipNr, NrFinal, LitFinală}

Descompunerea este salutară, deoarece rezolvă şi problema restricţiei de entitate; cea de-a doua relaţie va conţine numai codurile acordate la nivel de stradă, deci nici un atribut cheie nu va avea valori nule.

Relaţia STUDENŢI_EXAMENE

Senzaţia de déjà-vu care vă încearcă este îndreptăţită. Acesta este relaţia din paragraful 2.2 (figura 2.3) cu care am început să închinăm osanale normalizării. Este o relaţie simplă care se pretează la normalizare, manifestând redundanţele şi anomaliile la actualizare expuse pe scurt în paragrafele 2.2.1, 2.2.2, 2.2.3 şi 2.2.4.

Cheia primară fiind combinaţia (Matricol, CodDisc, DataExamen), automat sunt valabile următoatele DF:

(1) (Matricol, CodDisc, DataExamen) ⟶ NumePrenume

(2) (Matricol, CodDisc, DataExamen) ⟶ An

(3) (Matricol, CodDisc, DataExamen) ⟶ Specializare

(4) (Matricol, CodDisc, DataExamen) ⟶ DenumireDisc

(5) (Matricol, CodDisc, DataExamen) ⟶ NrCredite

(6) (Matricol, CodDisc, DataExamen) ⟶ Nota

STUDENŢI_EXAMENE ar fi în 2NF dacă nici una dintre aceste şase dependenţe nu ar fi parţială. Fără a le întrebuinţa prea serios, e suficient să ne reamintim că matricolul este desemnat să identifice în mod unic fiecare student. Dintr-un foc avem trei dependenţe:

(7) Matricol ⟶ NumePrenume

(8) Matricol ⟶ An

(9) Matricol ⟶ Specializare

Şi codul disciplinei are pretenţii de identificare fără ambiguitate a fiecărei „materii" parcurse de studenţi:

(10) CodDisc ⟶ DenumireDisc

(11) CodDisc ⟶ NrCredite

Din dependenţele (7)-(9) construim relaţia:

STUDENŢI {Matricol, NumePrenume, An, Specializare}

din (10) şi (11):

DISCIPLINE {CodDisc, DenumireDisc, NrCredite}

iar din relaţia iniţială rămâne:

EXAMENE {Matricol, CodDisc, DataExamen, Nota}

care nu are probleme cu valorile nule ale vreunui atribut cheie.

Avem toate motivele să fim mulţumiţi. Dacă ne raportăm la paragraful 2.2.1, observăm că redundanţele au fost eliminate. Pe de o parte, numele, specializarea şi anul de studiu ale fiecărui student apar acum o singură dată (în tuplul corespunzător studentului din relaţia STUDENŢI). Pe de altă parte, tot o singură dată apar şi denumirea unei discipline şi numărul de credite alocate acesteia (relaţia DISCIPLINE).

Lucrurile stau la fel de bine şi în ceea ce priveşte diminuarea anomaliilor manifestate la editarea datelor. Astfel, privitor la anomaliile de inserare (paragraful 2.2.2), din momentul înscrierii în anul I, până la cel al primului examen, studentul poate fi inserat în tabela STUDENŢI, urmând ca, o dată cu prima sa sesiune de examene, să-i fie introduse înregistrări în EXAMENE. Modificarea denumirii unei discipline sau a numărului de credite ori corectarea unei eventuale erori strecurate în numele studentului sau al specializării/anului, toate se fac într-un singur loc, pe tuplul corespunzător din STUDENŢI sau DISCIPLINE, ceea ce înseamnă că am scăpat şi de anomaliile din paragraful 2.2.3. În fine, paragraful 2.2.4 atrăgea atenţia asupra riscului pierderii unor informaţii valoroase la ştergerea unei linii din STUDENŢI_EXAMENE (un soi de aruncatul copilului o dată cu albia). Nici acest pericol nu mai este de actualitate în noua structură.

Relaţia VÂNZĂRI

Şi relaţia universală VÂNZĂRI încalcă preceptele relaţionalului (în fapt, am putea să-i spunem *tabelă*, pentru a respecta adevărul, tabela fiind asociată cu SQL, care este mult prea tolerant cu elemente ce fac dintr-o tabelă să nu fie o relaţie – unicitate, valori nule, lipsa cheii primare etc.), întrucât unul dintre atributele care asigură unicitate fiecărui tuplu din relaţia iniţială, CodÎnc (codul încasării), are valori nule pentru facturile „proaspăt" emise, de fapt, pentru toate facturile din care nu s-a încasat nici o leţcaie. Dacă nu s-ar fi pus problema restricţiei de entitate, combinaţia de atribute care diferenţiază cu siguranţă orice tuplu de celelalte este (NrFact, Linie, CodÎnc). Lucrurile nu sunt atât de grave, deoarece în setul de DF (vezi paragraful 4.1.2) apar destule surse care sunt subansambluri ale pseudocheii primare. Aşadar, să parcurgem cele 6 etape ale deja celebrului „îndrumar" de aducere a unei baze de date în 2FN.

1. Cheia primară a relaţiei este, după cum am afirmat nu mai sus cu câteva rânduri: (NrFact, Linie, CodÎnc).

2. Dependențele manifestate în relație sunt cele 54 din paragraful 4.1.2.

3. Dependențele (50), (51), (52) și (53) au drept sursă CodÎnc. Cu totul altfel stau lucrurile pentru NrFact. Dacă în paragraful 4.1.2 identificasem dependențele (37), (38), (39) și (40), am văzut în 4.1.3, când am discutat despre tranzitivitate, că acestora li se mai pot adăuga încă 12, adică (55)-(66). Am încheiat inventarierea DF în care sursele sunt simple și atribute cheie și continuăm cu surse compuse, alcătuite din două dintre cele trei atribute, la acest „capitol" având o singură dependență, (54), și cele de la (41) la (46).

4. Creăm noile relații.
Din (50), (51), (52) și (53) rezultă:

ÎNCASĂRI {CodÎnc, DataÎnc, CodDoc, NrDoc, DataDoc}

Din (37), (38), (39) și (40), plus (55)-(66) se obține:

FACTURI {NrFact, CodCl, DataFact, Obs, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost, Localitate, Comuna, Jud, Județ, Regiune}

Din dependența (54) se poate „izola" relația:

ÎNCAS_FACT {CodÎnc, NrFact, Tranșa}

Pe baza DF (41)-(46) se construiește:

LINII_FACTURI {NrFact, Linie, Cantitate, PrețUnit, CodPr, DenPr, UM, ProcTVA, Grupa}

5. Cu totul remarcabil este ceea ce mai rămâne din relația inițială: FACTURI_ÎNCASĂRI {NrFact, Linie, CodÎnc}.

6. În afara celor trei atribute cheie din relația inițială – nimic. Dacă în 1NF eludam problema restricției de entitate, pe motiv că în 2NF lucrurile se rezolvă, acum ne propunem să tranșăm discuția. Pentru ca să respecte restricția de entitate, relația FACTURI_ÎNCASĂRI nu poate „primi" tupluri, decât în momentul unei prime încasări a unei facturi. Dar, de fapt, ceea ce se încasează într-o tranșă nu este o linie (un produs) dintr-o factură, ci *o parte* sau *întreaga* factură, lucru pe care-l reflectă pe deplin dependența (54) și relația ÎNCAS_FACT. Prin urmare, această ultimă relație nu ne este de nici un folos, și, pur și simplu, renunțăm la ea. Senzația este oarecum ciudată, mai ales că un asemenea demers nu este fundamentat prea riguros în teoria atât de matematizată a normalizării.

În concluzie, în 2NF, relația universală VÂNZĂRI va avea structura relațiilor: ÎNCASĂRI, FACTURI, ÎNCAS_FACT și LINII_FACTURI.

Forme normalizate optimale

Dacă avem în vedere teorema lui Heath, ruperea unei relații în mai multe tabele poate fi făcută cu oarecare frenezie. Astfel, în LOCALITĂȚI_CODURI_VECHI, pe

baza DF: CodPoştal ⟶ OraşComună şi CodPoştal ⟶ Judeţ, baza de date poate fi adusă în 2NF şi astfel:

CODPOST_ORCOM {CodPostal, OraşComună}

CODPOST_JUD {CodPostal, Judeţ}

şi

SATE {CodPostal, Sat}

Este clar că avem de-a face cu o exagerare, deşi toate cele trei relaţii respectă cu sfinţenie condiţiile formei a doua normale. De cele mai multe ori, în normalizare se aplică principiul obţinerii celui mai mic număr de relaţii care să respecte condiţiile uneia sau alteia dintre formele normalizate.

CAPITOLUL 5

A treia formă normalizată şi forma normală Boyce-Codd

A treia formă normalizată (3NF) prezintă un interes aparte, deoarece este considerată de mulţi specialişti drept un minimum acceptabil pentru structura unei baze de date relaţionale. Bazată pe a doua formă normală şi pe dependenţele funcţionale tranzitive (de fapt, pe eliminarea lor), 3NF a cunoscut două definiţii majore, una „originală", formulată de Codd însuşi (cu variantele sale „folclorice"), şi a doua ce acoperă câteva lipsuri ale primei, numită, după autorii săi, *Boyce-Codd Normal Form*, în traducere oarecum liberă *forma normală Boyce-Codd*, mai pe românește, *BCNF*.

5.1. Dependenţe funcţionale directe/tranzitive

O dependenţă funcţională `Dată1` ⟶ `Dată2` este directă atunci când nu există o `Dată3` care angajează o dependenţă funcţională tranzitivă de genul:

`Dată1` ⟶ `Dată3` ⟶ `Dată2`.

Formal, dacă $A_i \longrightarrow A_j$ şi $A_j \longrightarrow A_k$, atunci $A_i \longrightarrow A_k$ şi se notează:

$$A_i \longrightarrow A_j \longrightarrow A_k.$$

Pentru relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 (figura 3.7), dependenţele (7), (8), (9) şi (10) din paragraful 4.1.2 sunt tranzitive, deoarece:

`Cota` ⟶ `ISBN` ⟶ `Titlu`

`Cota` ⟶ `ISBN` ⟶ `Editura`

`Cota` ⟶ `ISBN` ⟶ `LocSediuEd`

`Cota` ⟶ `ISBN` ⟶ `AnApariţie`

În plus, şi DF (13) este tranzitivă, întrucât:

`ISBN` ⟶ `Editura` ⟶ `LocSediuEd`

Pentru relaţia VÂNZĂRI, dependenţele tranzitive simple sunt mai numeroase:

`CodPost` ⟶ `Jud` ⟶ `Judeţ`

CodPost ⟶ Jud ⟶ Regiune
CodCl ⟶ CodPost ⟶ Localitate
CodCl ⟶ CodPost ⟶ Comuna
CodCl ⟶ CodPost ⟶ Jud ⟶ Judeţ
CodCl ⟶ CodPost ⟶ Jud ⟶ Regiune
NrFact ⟶ CodCl ⟶ DenCl
NrFact ⟶ CodCl ⟶ CodFiscal
NrFact ⟶ CodCl ⟶ StradaCl
NrFact ⟶ CodCl ⟶ NrStradaCl
NrFact ⟶ CodCl ⟶ BlocScApCl
NrFact ⟶ CodCl ⟶ TelefonCl
NrFact ⟶ CodCl ⟶ CodPost
NrFact ⟶ CodCl ⟶ CodPost ⟶ Localitate
...

dar şi

(NrFact, Linie) ⟶ CodPr ⟶ DenPr
(NrFact, Linie) ⟶ CodPr ⟶ UM
(NrFact, Linie) ⟶ CodPr ⟶ ProcTVA
(NrFact, Linie) ⟶ CodPr ⟶ Grupa

Pe baza noţiunilor prezentate, se poate defini *închiderea tranzitivă* a unui ansamblu de dependenţe funcţionale. *Închiderea tranzitivă* reprezintă ansamblul *dependenţelor totale directe + dependenţele totale tranzitive*. *Acoperirea minimală*, definită în capitolul anterior, este subansamblul minimal de dependenţe funcţionale elementare care permit determinarea tuturor celorlalte dependenţe funcţionale. Cu alte cuvinte: (1) nici o dependenţă funcţională nu este redundantă şi (2) orice dependenţă funcţională totală face parte din închiderea tranzitivă. Spre deosebire de ceea ce am discutat în capitolul anterior, acum suntem în măsură să eliminăm din ansamblul DF şi pe cele tranzitive.

Astfel, pentru relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3, acoperirea minimală este alcătuită din cinci DF:

- Cota ⟶ ISBN
- ISBN ⟶ Titlu
- ISBN ⟶ Editura
- ISBN ⟶ AnApariţie
- Editura ⟶ LocSediuEd

iar închiderea tranzitivă este următorul ansamblu de dependenţe:

- Cota ⟶ ISBN
- Cota ⟶ Titlu

- Cota ⟶ Editura
- Cota ⟶ LocSediuEd
- Cota ⟶ AnApariţie
- ISBN ⟶ Titlu
- ISBN ⟶ Editura
- ISBN ⟶ LocSediuEd
- ISBN ⟶ AnApariţie
- Editura ⟶ LocSediuEd

Acoperirea minimală poate fi uşor reprezentată prin graful DF. Dependenţele tranzitive sunt foarte uşor de identificat şi eliminat. Spre exemplu, dacă din figura 4.8 eliminăm săgeţile ce reprezintă drumul direct dintre două atribute/grupuri şi un atribut destinaţie, atât timp cât între respectivele atribute/conectori există un „traseu ocolitor", se obţine graful acoperirii minimale - vezi figura 5.1.

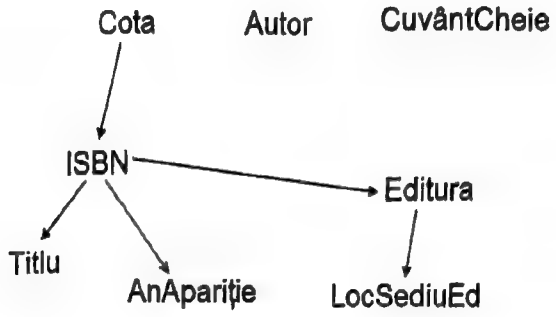


Figura 5.1. Graful DF de acoperire minimală pentru relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3

Este drept, atributele `Autor` şi `CuvântCheie` atârnă inutil în figură, stricând imaginea grafului. Le-am păstrat însă pentru a nu omite nici un atribut din relaţia universală şi a indica cheia primară a relaţiei.

Pentru baza de date VÂNZĂRI, acoperirea minimală, adică setul iniţial de DF din care au fost eliminate dependenţele simetrice (redundante), parţiale şi tranzitive, este următorul set de dependenţe funcţionale reprezentat în figura 4.9:

- Jud ⟶ Judeţ
- Jud ⟶ Regiune
- CodPost ⟶ Localitate
- CodPost ⟶ Comună
- CodPost ⟶ Jud
- CodCl ⟶ DenCl
- CodCl ⟶ CodFiscal
- CodCl ⟶ StradaCl
- CodCl ⟶ NrStradaCl
- CodCl ⟶ BlocScApCl
- CodCl ⟶ TelefonCl
- CodCl ⟶ CodPost

- CodPr ⟶ DenPr
- CodPr ⟶ UM
- CodPr ⟶ Grupa
- CodPr ⟶ ProcTVA
- NrFact ⟶ CodCl
- NrFact ⟶ DataFact
- NrFact ⟶ Obs
- (NrFact, Linie) ⟶ Cantitate
- (NrFact, Linie) ⟶ PreţUnit
- CodÎnc ⟶ DataÎnc
- CodÎnc ⟶ CodDoc
- CodÎnc ⟶ NrDoc
- CodÎnc ⟶ DataDoc
- (CodÎnc, NrFact) ⟶ Tranşă

5.2. Aducerea relaţiilor în a treia formă normală

Prima definiţie a celei de-a treia forme normale a unei relaţii poate fi formulată relativ simplu şi incremental, prin raportare la 2NF. O relaţie se află în 3NF dacă:

1. Se găseşte în 2NF.
2. Toate atributele care nu aparţin cheii primare nu depind funcţional de un alt atribut (ansamblu de atribute) care nu face parte din cheie.

A doua condiţie poate fi exprimată şi în maniera: *toate dependenţele funcţionale care leagă cheia primară de celelalte atribute sunt directe (netranzitive).*

O altă formulare a 3NF utilizează tot două condiţii:

- toate atributele non-cheie din R sunt independente unele faţă de altele, în sensul că nici unul dintre atributele neparticipante în cheia primară nu apare în vreo DF în care sursa este constituită dintr-o combinaţie de celelalte atribute non-cheie;
- toate atributele non-cheie din R sunt dependente ireductibil de cheia primară.

Important! Chris Date este poate singurul care, precaut, la începutul discuţiilor legate de primele trei forme normale, afirmă că situaţiile luate în considerare pornesc de la premisa că nu există mai multe chei candidate. Premisa este foarte importantă, întrucât atunci când nu se elimină dependenţele simetrice şi redundante pomenite în capitolul anterior, iar o relaţie are două ori mai multe chei candidate, sunt şanse mari să apară complicaţii. Vom reveni la sfârşitul acestui paragraf cu analiza bazei de date VÂNZĂRI şi vom vedea că şi precauţia lui Date este insuficientă, deoarece, în fapt, problema nu este numai a cheilor candidate ale unei relaţii, ci şi a dependenţelor simetrice şi redundante.

Trecerea din 2NF în 3NF presupune eliminarea DF tranzitive şi se poate face, pentru o relaţie, în următoarea manieră:

a) Se identifică toate atributele ce nu fac parte din cheie şi sunt surse ale unor dependenţe funcţionale.

b) Pentru toate atributele identificate la punctul a se constituie câte o relaţie în care cheie primară va fi atributul respectiv, iar celelalte atribute, destinaţiile din dependenţele considerate.

Operaţiile a şi b se repetă şi pentru relaţiile „proaspăt" obţinute prin decompoziţie. De fiecare dată, din relaţiile supuse descompunerii se elimină atributele destinaţie ale surselor non-cheie (sună bine, nu-i aşa?), păstrându-se atributele sursă, în vederea stabilirii legăturilor între tabele (cheile străine) şi, implicit, declarării restricţiilor referenţiale.

Relaţia LINII_ARTICOLE_CONTABILE

Relaţia LINII_ARTICOLE_CONTABILE din figura 4.1 a fost adusă în capitolul anterior în 2NF prin descompunerea în două relaţii, NOTE_CTB şi ARTICOLE_CTB, prezentate în figura 4.14.

NOTE_CTB {NotaContabilă, Data}

ARTICOLE_CTB {NotaContabilă, Operaţiune, ContDebitor, ContCreditor, Suma}

Întrucât nici una dintre cele două relaţii nu conţine dependenţe funcţionale tranzitive, în 3NF baza de date are aceeaşi structură ca şi în 2NF.

Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3

Relaţia prezentată în figura 3.7 (paragraful 3.3.2) a fost adusă în 2NF în paragraful 4.4, prin ruperea sa în CĂRŢI {Cota, ISBN, Titlu, Editura, LocSediuEd, AnApariţie} şi TITLURI_AUTORI_CUVINTECHEIE {Cota, Autor, CuvântCheie} (vezi figura 4.15).

Cea de-a doua este alcătuită exclusiv din atribute cheie, deci nu se pune în discuţie 3NF (adică relaţia este deja în 3NF). Altfel stau lucrurile în CĂRŢI. Relaţia este în 2NF şi conţine dependenţe tranzitive. Există două atribute din afara cheii primare, ISBN şi Editura, care sunt surse de DF, astfel încât dependenţele (7), (8), (9), (10) şi (13) sunt tranzitive. Trecerea în 3NF se realizează prin constituirea a două relaţii pe care o să le numim EDITURI şi TITLURI:

EDITURI {Editura, LocSediuEd}

şi

TITLURI {ISBN, Titlu, Editura, AnApariţie}

iar din CĂRŢI rămâne:

EXEMPLARE {Cota, ISBN}

În concluzie, în 3NF, baza de date BIBLIOTECĂ este alcătuită din patru tabele:

EDITURI {Editura, LocSediuEd}

TITLURI {ISBN, Titlu, Editura, AnApariţie}

EXEMPLARE {Cota, ISBN}

TITLURI_AUTORI_CUVINTECHEIE {Cota, Autor, CuvântCheie}

Conţinutul acestora este cel din figura 5.2.

EDITURI

Editura	LocSediuEd
Polirom	Iaşi

TITLURI

ISBN	Titlu	Editura	AnAparitie
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Polirom	2002
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Polirom	2001

EXEMPLARE

Cotă	ISBN
III-13421	973-683-889-7
III-13422	973-683-889-7
III-13423	973-683-889-7
III-10678	973-683-709-2
III-10679	973-683-709-2

TITLURI_AUTORI_CUVINTECHEIE (fragment)

Cota	Autor	CuvântCheie
III-13421	Marin Fotache	baze de date
III-13421	Marin Fotache	SQL
III-13421	Marin Fotache	proceduri stocate
III-13421	Marin Fotache	FoxPro
III-13421	Marin Fotache	formulare
III-13421	Marin Fotache	orientare pe obiecte
III-13421	Marin Fotache	client-server
III-13421	Marin Fotache	web
	...	

Figura 5.2. Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 în 3NF

Din păcate, ne rămâne pe conştiinţă ultima tabelă, TITLURI_AUTORI_CUVINTECHEIE, care conţine un grad ridicat de redundanţă a datelor, problemă rezolvată pe baza unui alt tip de dependenţe – multivaloare (vezi capitolul viitor).

Relaţia DOTARE_JUCĂRII_1

Relaţia DOTARE_JUCĂRII_1 (figura 3.14) a fost adusă în 2NF (paragraful 4.4) sub forma celor două relaţii, FAMILII şi JUCĂRII, din figura 4.16. Ambele relaţii sunt şi în 3NF, deoarece nici una nu prezintă vreo dependenţă tranzitivă.

Relaţiile universale LOCALITĂŢI_CODURI_VECHI, CODURI_ORAŞE şi STUDENŢI_EXAMENE

Nici una dintre relaţiile ORAŞE_COMUNE/SATE/CODURI_OR, CODURI_ADRESE/CODURI_LOCALITĂŢI şi STUDENŢI/DISCIPLINE/EXAMENE nu conţine dependenţe tranzitive, deci sunt şi în 3NF.

Relaţia CODURI_NOI_V3

Această relaţie obţinută în paragraful 3.4 (vezi figura 3.20), care are drept cheie primară atributul `Id` şi care nu a intrat în discuţie la 2NF, prezintă următoarele dependenţe:

(1) `Id` ⟶ `CodPoştal`

(2) `Id` ⟶ `Localitate`

(3) `Id` ⟶ `Strada`

(4) `Id` ⟶ `TipNr`

(5) `Id` ⟶ `NrIniţial`

(6) `Id` ⟶ `LitIniţială`

(7) `Id` ⟶ `NrFinal`

(8) `Id` ⟶ `LitFinală`

(9) `Id` ⟶ `Comuna`

(10) `Id` ⟶ `Judeţ`

dar şi

(11) `CodPoştal` ⟶ `Localitate`

(12) `CodPoştal` ⟶ `Comuna`

(13) `CodPoştal` ⟶ `Judeţ`

Pe baza DF (11), (12) şi (13), dependenţele (2), (9) şi (10) sunt tranzitive, iar relaţia se rupe în:

CODURI_LOC {`CodPostal`, `Localitate`, `Comuna`, `Judeţ`}

şi

CODURI_ADRESE2 {`Id`, `CodPoştal`, `Strada`, `TipNr`, `NrIniţial`, `LitIniţială`, `NrFinal`, `LitFinală`}

Relaţia VÂNZĂRI

Ultimul exemplu din pagraful 4.4 a adus relaţia universală VÂNZĂRI în a doua formă normalizată, prin ruperea acesteia în patru tabele: ÎNCAS_FACT, ÎNCASĂRI, LINII_FACTURI şi FACTURI. Dar, după cum am văzut în paragraful precedent, există o serie de dependenţe tranzitive. Să examinăm pe rând cele cinci relaţii.

ÎNCAS_FACT {`CodÎnc`, `NrFact`, `Tranşa`} are un singur atribut non-cheie, deci nici vorbă de DF tranzitive.

ÎNCASĂRI {`CodÎnc`, `DataÎnc`, `CodDoc`, `NrDoc`, `DataDoc`} nu prezintă nici un atribut în afara `CodÎnc` care să fie sursă de DF.

Cu LINII_FACTURI {NrFact, Linie, Cantitate, PreţUnit, CodPr, DenPr, UM, ProcTVA, Grupa} situaţia este diferită. Atributul non-cheie CodPr este determinant în o serie de dependenţe, (33)-(36), pe baza acestora construindu-se relaţia PRODUSE {CodPr, DenPr, UM, ProcTVA, Grupa}; din LINII_FACTURI se elimină toate destinaţiile celor patru DF, obţinându-se LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit}.

În tabela FACTURI {NrFact, CodCl, DataFact, Obs, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost, Localitate, Comuna, Jud, Judeţ, Regiune} lucrurile sunt şi mai interesante:

- mai întâi, se cuvine de observat că Jud este sursă a DF (1) şi (3), aşa încât FACTURI se descompune în:
  - JUDEŢE {Jud, Judeţ, Regiune};
  - FACTURI_1 {NrFact, CodCl, DataFact, Obs, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost, Localitate, Comună, Jud};
- dar şi FACTURI_1 prezintă DF; în dependenţele (5), (6) şi (7), CodPost este sursă, aşa încât FACTURI_1 se descompune în:
  - CODURI_POŞTALE {CodPost, Localitate, Comună, Jud};
  - FACTURI_2 {NrFact, CodCl, DataFact, Obs, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost};
- în FACTURI_2 a mai rămas un singur atribut non-cheie, care este sursă în DF (8)-(16), CodCl, astfel încât şi aceasta se descompune în:
  - CLIENŢI {CodCl, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost};
  - FACT {NrFact, CodCl, DataFact, Obs}.

În concluzie, în 3NF, relaţia iniţială VÂNZĂRI este „spartă" în următoarele opt relaţii:

ÎNCAS_FACT {CodÎnc, NrFact, Tranşa}

ÎNCASĂRI {CodÎnc, DataÎnc, CodDoc, NrDoc, DataDoc}

PRODUSE {CodPr, DenPr, UM, ProcTVA, Grupa}

LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit}

JUDEŢE {Jud, Judeţ, Regiune}

CODURI_POŞTALE {CodPost, Localitate, Comună, Jud}

CLIENŢI {CodCl, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost}

FACT {NrFact, CodCl, DataFact, Obs}

Cheile candidate şi influenţa lor asupra 3NF

Promiteam în deschiderea paragrafului câteva detalii despre implicaţiile cheilor alternative asupra aducerii unei relaţii în 3NF, implicaţii prea puţin evocate în

literatura de profil. Astfel, revenim la relaţia 2NF LINII_FACTURI {NrFact, Linie, Cantitate, PreţUnit, CodPr, DenPr, UM, ProcTVA, Grupa} din baza de date VÂNZĂRI. Dacă luăm în consideraţie că şi denumirea produsului este unică, altfel spus, nu există două sortimente cu nume absolut identic, rezultă o serie paralelă de dependenţe (după cum am discutat în paragraful 4.1.4):

CodPr ⟶ DenPr	DenPr ⟶ CodPr
CodPr ⟶ UM	DenPr ⟶ UM
CodPr ⟶ Grupa	DenPr ⟶ Grupa
CodPr ⟶ ProcTVA	DenPr ⟶ ProcTVA

Dacă urmăm algoritmul prescris de trecere a relaţiei LINII_FACTURI din 2NF în 3NF, ar trebui să constituim câte o relaţie pentru fiecare sursă de DF care nu face parte din cheia primară a relaţiei, iar în relaţia ce rămâne din LINII_FACTURI (LINII_FACT_2) să păstrăm pe post de chei străine atât CodPr, cât şi DenPr:

PRODUSE_1 {CodPr, DenPr, UM, ProcTVA, Grupa}

PRODUSE_2 {DenPr, CodPr, UM, ProcTVA, Grupa}

LINII_FACT_2 {NrFact, Linie, CodPr, DenPr, Cantitate, PreţUnit}

Gogomănia este evidentă, însă, ca în atâtea alte cazuri, fundamentată ştiinţific! Iar dacă demonstraţia nu a fost suficient de impresionantă, ce ziceţi de relaţia FACTURI în 2FN, în care orice client poate fi identificat fără ambiguitate atât de CodCl, cât şi de DenCl şi de CodFiscal.

CodCl ⟶ DenCl	DenCl ⟶ CodCl	CodFiscal ⟶ CodCl
CodCl ⟶ CodFiscal	DenCl ⟶ CodFiscal	CodFiscal ⟶ DenCl
CodCl ⟶ StradaCl	DenCl ⟶ StradaCl	CodFiscal ⟶ StradaCl
CodCl ⟶ NrStradaCl	DenCl ⟶ NrStradaCl	CodFiscal ⟶ NrStradaCl
CodCl ⟶ BlocScApCl	DenCl ⟶ BlocScApCl	CodFiscal ⟶ BlocScApCl
CodCl ⟶ TelefonCl	DenCl ⟶ TelefonCl	CodFiscal ⟶ TelefonCl
CodCl ⟶ CodPost	DenCl ⟶ CodPost	CodFiscal ⟶ CodPost
CodCl ⟶ Localitate	DenCl ⟶ Localitate	CodFiscal ⟶ Localitate
CodCl ⟶ Comuna	DenCl ⟶ Comuna	CodFiscal ⟶ Comuna
CodCl ⟶ Jud	DenCl ⟶ Jud	CodFiscal ⟶ Jud
CodCl ⟶ Judeţ	DenCl ⟶ Judeţ	CodFiscal ⟶ Judeţ
CodCl ⟶ Regiune	DenCl ⟶ Regiune	CodFiscal ⟶ Regiune

Aşa că din LINII_FACTURI obţinem în prima fază a trecerii la 3NF nu mai puţin de patru relaţii, apoi continuăm cu descompunerea celor trei constituite pe baza seriilor de dependenţe de mai sus ş.a.m.d.

Soluţia este cea pe care am sugerat-o în capitolul 4. Pentru identificarea oricărei entităţi (produs, client, factură) păstrăm doar cheia primară (practic, renunţăm la celelalte chei candidate), iar toate dependenţele ce decurg din calitatea de cheie alternativă nu sunt luate în considerare.

5.3. Aducerea relaţiilor în 3NF prin grafuri şi matrice ale DF

Modul în care a fost prezentată transformarea succesivă a relaţiei universale iniţiale în 1NF, 2NF şi 3NF prin *descompuneri succesive* are certe valenţe pedagogice şi aplicabilitate practică. Descompunerea nu este însă singura manieră de normalizare a unei baze de date relaţionale.

Philip A. Bernstein este autorul unui model de normalizare prin *sinteza* unor relaţii binare (construite pe baza DF) în relaţii mai mari şi, astfel, aducerea bazei în 3NF[1]. În plus, autorul a demonstrat că schema obţinută conţine un număr minim de relaţii. Algoritmul este menţionat în majoritatea cărţilor dedicate bazelor de date, deşi nu toţi autorii îl descriu complet. Iată paşii:

1. Se elimină atributele exterioare din sursele tuturor dependenţelor funcţionale (setul F de DF). Un atribut este extern dacă eliminarea sa nu alterează închiderea lui F. Prin eliminarea din F se obţine un alt set de DF, notat G.
2. Se determină acoperirea neredundantă H din setul de DF notat G.
3. Se partiţionează H în grupuri, astfel încât toate DF dintr-un grup să aibă surse identice.
4. Se fuzionează cheile echivalente. Fie J = O. Pentru toate perechile de grupuri Hi şi Hj, cu sursele X, respectiv Y, se va realiza fuziunea H1 şi H2 în cazurile în care se manifestă bijectivitatea $X \leftrightarrow Y$ în H+. Pentru fiecare asemenea bijectivitate, se adaugă în J dependenţele $X \rightarrow Y$ şi $Y \rightarrow X$. Pentru orice $A \in Y$, dacă există dependenţa $X \rightarrow A$ în H, aceasta se şterge din H. Analog se procedează pentru fiecare dependenţă $Y \rightarrow B$ din H, unde $B \in X$.
5. Se elimină DF tranzitive. Se identifică seturi $H' \subseteq H$ care îndeplinesc condiţiile $(H' + J)+ = (H + J)+$ şi nici un subset din H' nu îndeplineşte această condiţie. Se adaugă fiecare DF din J în grupul său corespondent din H'.
6. Pentru fiecare grup, se construieşte o relaţie ce conţine toate atributele grupului respectiv. Atributele din sursa DF vor alcătui cheia primară a relaţiei respective.

Deşi s-a demonstrat că algoritmul este corect, aplicarea sa efectivă ridică destule dificultăţi, deoarece, după cum am mai discutat, închiderea şi acoperirea minimală ale seturilor de DF chiar şi pentru relaţii nu foarte voluminoase sunt greu de manevrat. Chris Date reproşează modelului (şi Bernstein acceptă observaţia) că operaţiunile executate în cadrul algoritmului de sinteză sunt exclusiv de natură sintactică şi nu iau în considerare latura semantică[2]. Însă exemplul luat de Date pentru a pune în dificultate algoritmul de sinteză constituie o capcană şi pentru varianta descompunerii.

O variantă mult mai directă şi relevantă pentru practicieni aparţine lui Henry Smith, care a publicat în *Communications of the ACM* un foarte interesant articol în care arată că o structură deplin normalizată poate fi obţinută pe baza unei riguroase

1. [Bernstein 76].
2. [Date 00], pp. 377-378.

diagrame de DF[3]. De fapt, diagramele sale seamănă mai degrabă cu grafuri decât cu forma lor consacrată[4]. Smith afirmă că descompunerea clasică, pornind de la relaţia universală, este greoaie, iar construirea relaţiei universale reclamă, de multe ori, un efort important. Lucrarea lui Smith este cu atât mai tentantă, cu cât nu operează cu algoritmi pentru determinarea închiderilor şi acoperirilor de seturi de DF. Ţinând seama că, în 3NF, relaţiile nu trebuie să conţină dependenţe parţiale şi tranzitive, se va reprezenta grafic acoperirea minimală.

De fapt, un avantaj decisiv al grafurilor DF este posibilitatea identificării imediate şi eliminării atât a dependenţelor parţiale, cât şi a celor tranzitive. În plus, după cum vom vedea în paragraful 5.5.2, există situaţii în care normalizarea „clasică" generează probleme, în timp ce, folosind graful, schema obţinută este net superioară.

Baza de date BIBLIOTECĂ

În capitolul 3 am demarat discuţiile legate de normalizarea bazei de date BIBLIOTECĂ, pentru care am convenit, nu fără strângere de inimă, că cea mai acceptabilă primă formă normală ar fi BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 (vezi figura 3.7). Pe baza dependenţelor identificate în capitolul 4, graful acoperirii minimale pentru universala BIBLIOTECĂ este reprezentat în figura 5.1. Totuşi, pentru ilustrarea modului de identificare şi eliminare a dependenţelor parţiale şi tranzitive pornim de la graful din figura 4.8. care va fi reprezentat în stânga figurii 5.3. Spre deosebire de figura „sursă", săgeţile ce reprezintă dependenţe parţiale sau tranzitive sunt punctate. Eliminându-le, obţinem graful din dreapta figurii, pe care delimităm câte o relaţie pentru fiecare sursă, simplă sau compusă. Relaţia corespondentă va conţine sursa, drept cheie primară, plus toate destinaţiile sale directe (şi totale).

În final, nu ne mai rămâne decât să dăm câte un nume fiecăreia dintre cele patru relaţii: EDITURI, TITLURI, EXEMPLARE, TITLURI_AUTORI_CUVINTECHEIE.

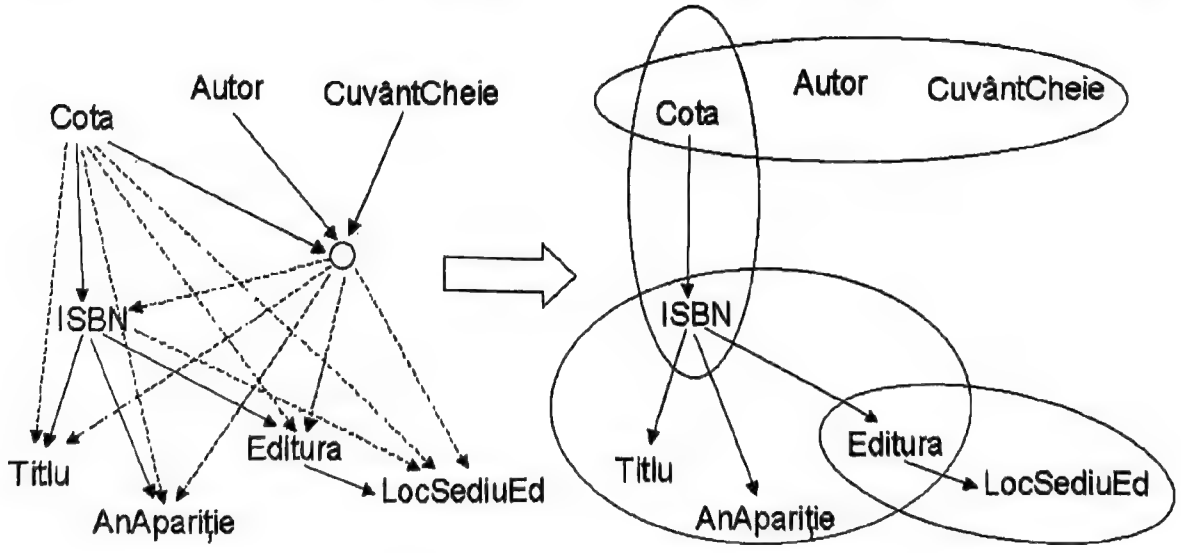


Figura 5.3. Cele patru tabele în 3NF pentru relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3

3. [Smith 85].
4. Vezi, spre exemplu, [Date 86] sau [Pratt & Adamski 91].

Relaţia STUDENŢI_EXAMENE

Cele 11 dependenţe identificate în paragraful 4.4 pot fi lesne reprezentate ca un graf – vezi figura 5.4. Dependenţele punctate sunt parţiale, deoarece destinaţiile lor sunt legate de surse simple, aşa că în final sunt eliminate din calcul.

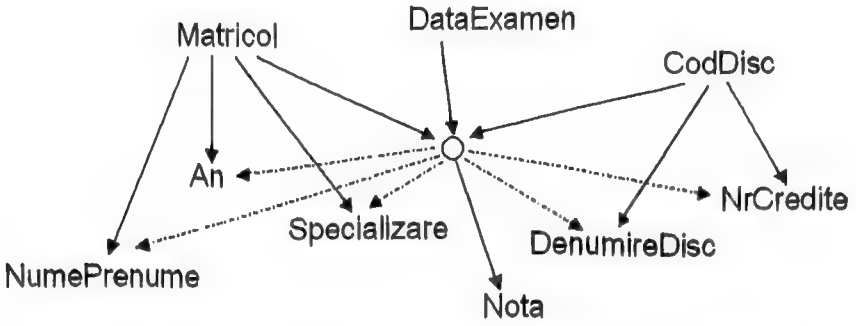


Figura 5.4. Aducerea relaţiei STUDENŢI_EXAMENE în 3NF cu ajutorul grafului DF

Pe graf sunt trei surse, aşa încât vom obţine trei relaţii, cu următoarele atribute: {Matricol, NumePrenume, An, Specializare}, {CodDisc, DenumireDisc, NrCredite} şi {Matricol, CodDisc, DataExamen, Nota}. Schema seamănă leit cu cea din paragraful 4.4.

Baza de date VÂNZĂRI

Prin reprezentarea sub formă de graf, economia de timp pentru aducerea în 3NF este şi mai spectaculoasă, după cum reiese şi din figura 5.5. Singura problemă ţine

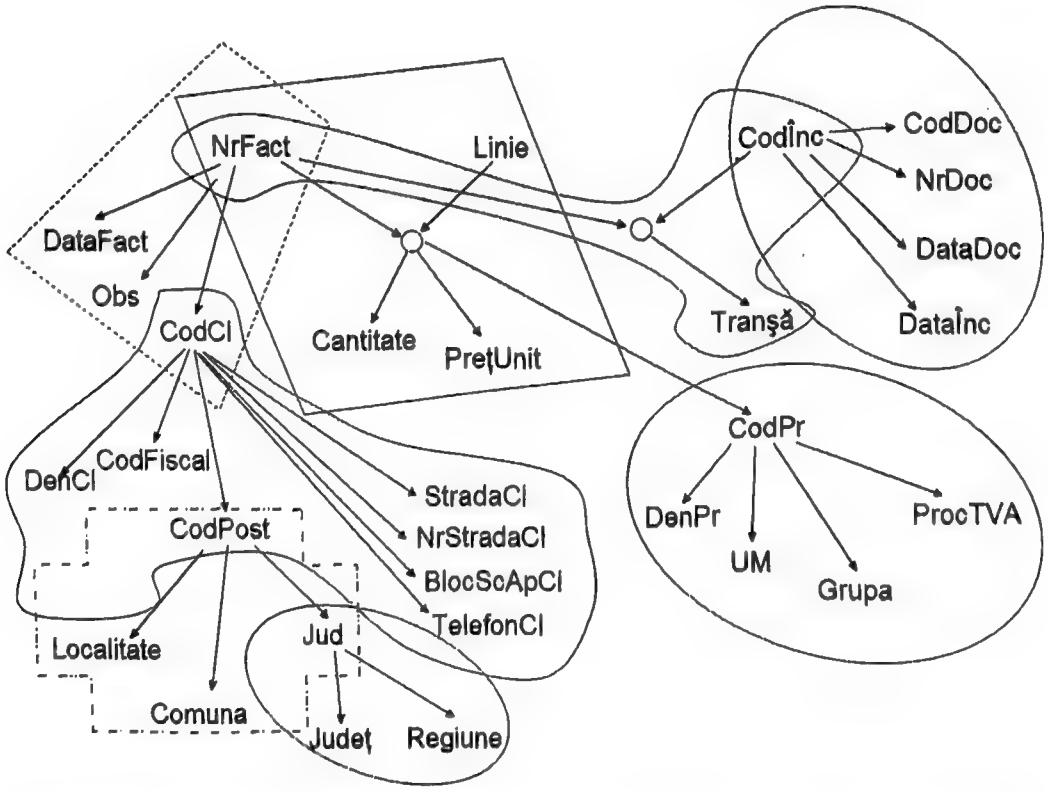


Figura 5.5. Decuparea relaţiilor în 3NF pentru baza de date VÂNZĂRI

de atenţia în delimitarea tuturor surselor şi dependenţelor lor, ceea ce poate crea oarecare confuzie.

Pentru majoritatea cazurilor întâlnite în practică, construirea grafului dependenţelor funcţionale ce alcătuiesc acoperirea minimală este una dintre cele mai simple şi la îndemână modalităţi de aducere a bazei de date în 3NF. Într-o manieră asemănătoare se pot determina relaţiile celei de-a treia forme normale, utilizând matricea dependenţelor incluse în acoperirea minimală.

5.4. Forma normală Boyce-Codd

Definirea celei de-a treia forme normale, aşa cum a fost formulată iniţial de Codd, ridica o serie de probleme în anumite circumstanţe, probleme pe care formularea din acest paragraf le rezolvă. Chris Date este de părere că situaţiile în care definiţia originală a 3NF este insuficientă se traduc prin trei condiţii simultane[5]:

- relaţia are mai multe chei candidate;
- o parte dintre cheile candidate sunt compuse;
- cheile compuse au atribute în comun.

Întrucât îndeplinirea simultană a celor trei condiţii se întâlneşte destul de rar în practică, a treia formă normală, aşa cum a fost formulată în paragraful precedent, este suficientă. Se spune despre o relaţie că este în forma normală Boyce-Codd (BCNF) dacă:

1. se află deja în 3NF;
2. nu există nici o DF a cărei sursă să fie un atribut necomponent al cheii, iar destinaţia, un atribut din cheie.

Într-o altă formulare, o relaţie R este în BCNF dacă şi numai dacă *orice determinant (sursă de dependenţă funcţională* sau *supercheie) este o cheie candidată*. Această exprimare are şi avantajul de a nu lega BCNF de 3NF, unii autori preferând să discute problema normalizării direct în termenii BCNF, fără a trece prin formele intermediare[6]. Numele formei normalizate este discutabil, deoarece primul care elaborează o definiţie echivalentă a ceea ce reprezintă astăzi BCNF este Ian Heath. În penultima ediţie a cărţii sale ce a împlinit recent 25 de ani, Chris Date afirmă că ar fi fost mai indicată titulatura *forma normală a lui Heath* (*Heath Normal Form*)[7].

Cele trei condiţii formulate de Chris Date au fost preluate şi în alte lucrări, precum cele scrise de Rebecca M. Riordan[8] sau, la noi, de Dumitru Oprea[9] şi de

5. [Date 86], p. 374.
6. Vezi, spre exemplu, [Garcia-Molina ş.a. 02].
7. [Date 00], p. 366.
8. [Riordan 99], p. 38.
9. [Oprea 99], p. 351.

Robert Dollinger[10]. Este drept că Date, precaut, afirmă că suprapunerea (cel puţin un atribut în comun) cheilor candidate nu atrage întotdeauna necesitatea transformării relaţiei în BCNF[11].

Primul contraexemplu

Curios lucru însă, din exemplele discutate în ultimele capitole putem servi un contraexemplu – tabela LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit} pe care am obţinut-o prin normalizare în paragraful anterior:

- tabela are două chei candidate, (NrFact, Linie) şi (NrFact, CodPr);
- cele două surse de DF (superchei) sunt compuse;
- atributul NrFact este comun ambelor superchei.

Toate cele trei condiţii sunt îndeplinite; cu toate acestea, relaţia LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit} este şi în 3NF, şi în BCNF, deoarece ambii determinanţi sunt chei candidate ale relaţiei.

Exemplificarea relaţiilor aflate în 3NF, dar nu şi în BCNF este destul de zgârcită şi confuză în literatura de specialitate[12]. Rebecca Riordan ia în discuţie, după modelul lui Date[13], următoarea relaţie[14]: R {CodFurnizor, NumeFurnizor, CodProdus, Cantitate, PreţUnitar}. Cele două chei candidate ale lui R sunt (CodFurnizor, CodProdus) şi (NumeFurnizor, CodProdus). Fiind compuse şi având un atribut comun, cazul pare ideal pentru aducerea în BCNF.

Fără a dramatiza prea mult lucrurile, mai întâi cred că exemplul nu prea are relevanţă practică, deoarece datele despre produsele achiziţionate sunt preluate din facturi, nefiind legate de furnizori decât la raportare (agregări, analize ale datelor). Mult mai coerentă este structura: FP {CodFurnizor, NumeFurnizor, NrFactura, DataFact, ValoareTotală}.

Trecând însă peste „legitimitatea" relaţiei, problema are rezolvare foarte simplă. Codul identifică fără ambiguitate un furnizor, deci CodFurnizor ⟶ NumeFurnizor. Este drept că şi reciproca este valabilă, dar dacă eliminăm dependenţele simetrice, avem de-a face în R cu o dependenţă funcţională (CodFurnizor, CodProdus ⟶ NumeFurnizor) parţială. Prin urmare, relaţia R poate fi descompusă încă din 2NF în R1 {CodFurnizor, NumeFurnizor} şi R2 {CodFurnizor, CodProdus, Cantitate, PreţUnitar}.

10. [Dollinger 98], p. 160. Dollinger preia numai parţial ideea lui Date, dar greşeşte afirmând că definiţiile 3NF (formularea „clasică") şi BCNF sunt echivalente numai atunci când relaţia R supusă normalizării are o singură cheie.
11. [Date 86], p. 378.
12. În [Lungu ş.a. 95], p. 170 şi pp. 184-185, precum şi în [Băscă 97], pp. 110-111, exemplele sunt strict teoretice, iar în [Florescu ş.a. 99], p. 85, exemplul relaţiei în BCNF este neexplicat şi inexplicabil.
13. [Date 86], p. 376.
14. [Riordan 99], pp. 38-39.

Relaţia PROIECTE

Candace Fleming şi Barbara von Halle exemplifică utilitatea BCNF prin relaţia PROIECTE din figura 5.6[15].

PROIECTE

ProiectNr	Membru	Supervizor
1	John	Oprea
1	Cătălin	Marin
1	Gabi	Oprea
2	John	Airinei
2	Cătălin	Airinei
3	Mircea	Luminiţa
3	Doina	Fătu
3	John	Fătu

Figura 5.6. Relaţie aflată în 3NF, nu însă şi în BCNF

Restricţii:
- la fiecare proiect există mai mulţi supervizori şi mai mulţi membri;
- un membru poate participa la oricâte proiecte;
- o persoană poate superviza un singur proiect;
- un supervizor poate coordona (în cadrul aceluiaşi proiect) mai mulţi membri;
- într-un proiect, un membru este coordonat de un singur supervizor.

În aceste condiţii, se verifică următoarele DF:

(1) (`ProiectNr, Membru`) ⟶ `Supervizor`

(2) (`Supervizor, Membru`) ⟶ `ProiectNr`

Prin urmare, relaţia are două chei candidate. Totodată însă, există şi DF `Supervizor` ⟶ `ProiectNr`, ceea ce face din dependenţa (2) una parţială. Atributul `Supervizor` este determinant fără a fi cheie candidată. În aceste condiţii, relaţia *nu* este în BCNF. Pentru a o aduce în această formă normalizată este necesară transformarea

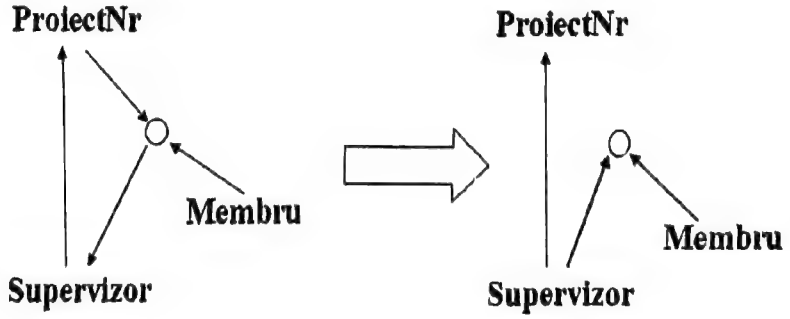


Figura 5.7. Transformarea grafului DF pentru relaţia PROIECTE

15. Exemplul este preluat şi în alte lucrări, precum [Teorey 99].

grafului DF, aşa cum arată figura 5.7 (pentru discuţii privind grafurile DF în condiţiile BCNF, vezi şi paragraful 5.5.5).

Cele două relaţii obţinute sunt:

SUPERVIZORI {Supervizor, ProiectNr}
şi
MEMBRI {Supervizor, Membru}

reprezentate în figura 5.8.

SUPERVIZORI

Supervizor	ProiectNr
Oprea	1
Marin	1
Airinei	2
Luminița	3
Fătu	3

MEMBRI

Supervizor	Membru
Oprea	John
Marin	Cătălin
Oprea	Gabi
Airinei	John
Airinei	Cătălin
Luminița	Mircea
Fătu	Doina
Fătu	John

Figura 5.8. Aducerea relației PROIECTE în BCNF

O primă problemă semnalată de cele două autoare se referă la pierderea unei restricţii de integritate: *la un proiect, un membru are un singur supervizor*. Cu alte cuvinte, noua structură a BD permite ca unui membru să-i fie asociaţi doi supervizori în cadrul aceluiaşi proiect, ceea ce este incorect în condiţiile date. În plus, exemplului prezentat i se poate reproşa şi rigiditatea, deoarece în decursul anilor un supervizor poate răspunde de mai multe proiecte, dar acest din urmă neajuns este mai degrabă unul mărunt.

Nişte profesori, cursuri şi specializări

Cred că, de la C.J. Date încoace, cel mai frecvent întâlnit exemplu pentru ilustrarea formei normale Boyce-Codd este cel privitor la cursuri-profesori-studenţi. Nu vom face rabat de la „regulă" şi definim tabela SCP, reprezentată în figura 5.9, în condiţiile în care *un profesor poate preda la oricâte specializări, dar numai un singur curs* (asta da profilare)!

SCP

Specializare	Curs	Profesor
Contabilitate	Bazele informaticii economice	Fătu
Finanțe	Bazele informaticii economice	Fătu
Contabilitate	Proiectarea sistemelor informatice	Oprea
Contabilitate	Baze de date	Airinei
Finanțe	Monedă	Turliuc
Statistică	Bazele informaticii economice	Fotache
Finanțe	Baze de date	Filip

Figura 5.9. Relația SCP

Să analizăm dependenţele funcţionale.

- un profesor predă un singur curs:

  `Profesor` ⟶ `Curs`

- un profesor predă la mai multe specializări:

  `Profesor` —/→ `Specializare`

- un acelaşi curs poate fi ţinut de mai mulţi profesori:

  `Curs` —/→ `Profesor`

- un acelaşi curs poate fi în planul de învăţământ al mai multor specializări:

  `Curs` —/→ `Specializare`

- la o specializare, în planul de învăţământ sunt incluse mai multe cursuri (din păcate):

  `Specializare` —/→ `Curs`

- la o specializare predau cursuri mai mulţi profesori:

  `Specializare` —/→ `Profesor`

- la o specializare, un curs este predat de un singur profesor:

  (`Specializare`, `Curs`) ⟶ `Profesor`

- un profesor poate ţine cursul la mai multe specializări:

  (`Profesor`, `Curs`) —/→ `Specializare`

Cuplul de atribute (`Specializare`, `Curs`) este cheie primară. Pe baza dependenţei funcţionale `Profesor` ⟶ `Curs` în BCNF, tabela SCP va fi descompusă în două tabele, SP şi PC, prezentate în figura 5.10.

SP

Specializare	Profesor
Contabilitate	Fătu
Finanţe	Fătu
Contabilitate	Oprea
Contabilitate	Airinei
Finanţe	Turliuc
Statistică	Fotache
Finanţe	Filip

PC

Profesor	Curs
Fătu	Bazele informaticii economice
Oprea	Proiectarea sistemelor informatice
Airinei	Baze de date
Turliuc	Monedă
Fotache	Bazele informaticii economice
Filip	Baze de date

Figura 5.10. Tabelele SP şi PC aflate în BCNF

Relaţia TELEFOANE

Un exemplu sugestiv de relaţie aflată în 3NF, dar nu şi în BCNF l-am întâlnit în lucrarea lui Robert Dollinger[16], care ia în discuţie relaţia TELEFOANE {`Oraş`,

16. [Dollinger 98], pp. 160-161.

Judeţ, Prefix), ce conţine prefixele telefonice ale tuturor oraşelor (şi, eventual, comunelor ce au centrală automată) din ţară.

Tabela are două chei candidate, (Oraş, Prefix) şi (Oraş, Judeţ):

(1) (Oraş, Prefix) ⟶ Judeţ

(2) (Oraş, Judeţ) ⟶ Prefix

Iată care sunt premisele autorului:

- în ţară există mai multe comune cu acelaşi nume, plasate în judeţe diferite:

  Oraş —/→ Prefix

  Oraş —/→ Judeţ

  Prefix —/→ Oraş

- un prefix nu poate fi alocat pentru localităţi din mai multe judeţe:

  (3) Prefix ⟶ Judeţ

În condiţiile în care cheia primară este combinaţia (Oraş, Judeţ), în relaţie există un atribut din afara cheii (Prefix, care altul?) în postura de sursă a unei DF în care destinaţia este un atribut din cheie, lucru pus în evidenţă de figura 5.11.

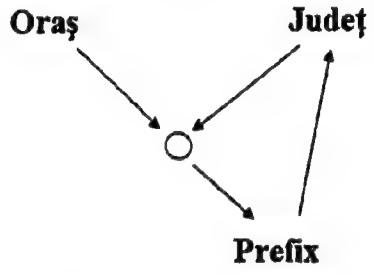


Figura 5.11. Graful DF pentru relaţia TELEFOANE

Graful din figura 5.11 nu este unul complet. Pe bună dreptate, întrucât forma din figura 5.12 nu este una dezirabilă, deoarece dependenţa Prefix ⟶ Judeţ face din (Oraş, Prefix) ⟶ Judeţ una parţială, care trebuie deci eliminată.

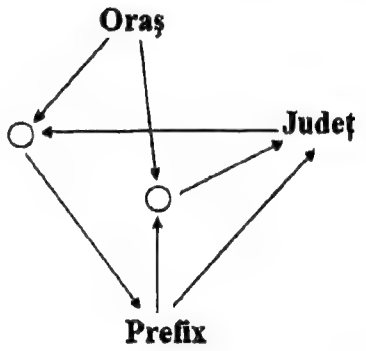


Figura 5.12. Graful complet al dependenţelor relaţiei TELEFOANE

Conformitatea cu regulile BCNF este realizată prin spargerea relaţiei TELEFOANE în:

PREFIXE_JUD {Prefix, Judeţ}

şi

ORCOM_PREFIX {Prefix, Oraş}

S-ar cuveni câteva discuţii vizavi de modul de tratare a acestui exemplu de către autor. Mai întâi, Dollinger trece sub tăcere faptul că există o dependenţă simetrică pentru (3), şi anume:

(4) Judeţ ⟶ Prefix

deoarece pe tot cuprinsul unui judeţ operează un singur prefix. Pierderea, prin descompunere, a dependenţei (2), (Oraş, Judeţ) ⟶ Prefix, este totuşi nesemnificativă, atât timp cât aceasta, datorită (4), este una parţială.

Consecvenţi discuţiilor din capitolul precedent vizavi de dependenţele simetrice, eliminăm una dintre dependenţele (3) şi (4). Rezultatul nu suferă modificări, întrucât dependenţa (3) o face pe (1) parţială, în timp ce (4) o face pe (2) parţială.

Lucrări de diplomă

Un exemplu sugestiv de relaţie aflată în 3NF ce întruneşte condiţiile de a fi convertită în BCNF este STUD_DOMENIU_CONDUCĂTOR, propusă de Dumitru Oprea[17]. Relaţia gestionează informaţiile legate de lucrările de diplomă ale studenţilor care şi-au propus să absolvească o facultate: {Matricola_Student, Domeniu, Conducător}. Deşi nu chiar reală, situaţia aleasă este cea în care un student îşi poate alege la diplomă unul, două, sau chiar mai multe domenii, de fiecare domeniu fiind responsabili o serie de profesori[18]. Cheia primară este combinaţia (Matricola_Student, Domeniu).

În condiţiile în care:

- un student îşi poate alege o temă din mai multe domenii;
- la fiecare domeniu, un student „beneficiază" de un singur profesor conducător;
- fiecare domeniu este acoperit de mai mulţi conducători şi
- fiecare conducător coordonează un singur domeniu.

relaţia are două chei candidate, (Matricola_Student, Domeniu) şi (Matricola_Student, Conducător), iar Conducător ⟶ Domeniu. Aducerea relaţiei în BCNF presupune descompunerea sa în

CONDUCĂTORI_DOMENII {Conducător, Domeniu} şi

STUDENŢI_CONDUCĂTORI {Matricola_Student, Conducător}

17. [Oprea 99], pp. 351-352.
18. Situaţia nu este chiar reală, deoarece ar însemna ca o lucrare de diplomă să fie coordonată de doi, sau chiar mai mulţi profesori din domenii diferite. Or, după cum ştiţi, profesorii nu se prea pot înghiţi unii pe alţii.

Al doilea contraexemplu BCNF

La începutul acestui paragraf, prezentam cele trei condiţii care, îndeplinite, atrag necesitatea aducerii relaţiei în BCNF. Reluăm tabela SCP, dar schimbăm ipoteza de lucru: *un profesor poate ţine mai multe cursuri, dar fiecare curs îl predă la o singură specializare, iar la o specializare orice curs este ţinut de un singur profesor*, noua relaţie fiind denumită SCP2 – vezi figura 5.13.

SCP2

Specializare	Curs	Profesor
Contabilitate	Bazele informaticii economice	Fătu
Finanţe	Bazele informaticii economice	Filip
Contabilitate	Proiectarea sistemelor informatice	Oprea
Contabilitate	Baze de date	Airinei
Finanţe	Monedă	Turliuc
Statistică	Bazele informaticii economice	Fotache
Statistică	Bazele informaticii economice	Airinei
Finanţe	Baze de date	Filip

Figura 5.13. Tabela SPC2

În aceste noi condiţii, singurele DF sunt:

(`Profesor, Curs`) ⟶ `Specializare`

(`Specializare, Curs`) ⟶ `Profesor`

Deoarece un profesor poate preda două cursuri la o aceeaşi specializare,

(`Profesor, Specializare`) —/→ `Curs`

Tabela SCP2 îndeplineşte cele trei condiţii:
- relaţia are două chei candidate: (`Profesor, Curs`) şi (`Specializare, Curs`);
- cele două chei candidate sunt compuse;
- cheile compuse au atributul `Curs` în comun.

Aceasta ar însemna că SCP este în 3NF şi trebuie adusă în BCNF. Totuşi, relaţia SCP2 este deja în BCNF, deoarece, conform definiţiei, *oricare determinant* (în SCP2 sunt doi determinanţi) *este cheie candidată a relaţiei*.

5.5. Câteva probleme ale normalizării

Aproape toţi autorii importanţi atrag atenţia asupra faptului că, deşi riguros formalizată, normalizarea nu este un panaceu. Există numeroase reguli semantice pe care normalizarea nu le poate încorpora. În plus, o bază de date excesiv normalizată poate crea probleme la implementare, cum ar fi o viteză de acces inacceptabilă. Multe situaţii trebuie tratate cu precauţie, iar uneori intuiţia este preferabilă rigidităţii unui formalism.

Dacă în capitolul anterior discutam despre dependenţe simetrice, prin eliminarea cărora scăpam de multe complicaţii, în cele ce urmează ne vom ocupa de alte aspecte semantice ale normalizării, care sunt „alunecoase", mai uşor de identificat sau chiar insolubile.

5.5.1. *DF aparent tranzitive, dar care conţin informaţii neredundante*

Să luăm un exemplu foarte simplu: R {Profesor, Birou, Catedră, Telefon}, care conţine date despre „domiciliul" în facultate al fiecărui profesor.

- Un profesor are un singur birou (Birou identifică fără ambiguitate sala respectivă):

  (1) Profesor ⟶ Birou

- Un profesor face parte dintr-o singură catedră:

  (2) Profesor ⟶ Catedră

- Într-un birou pot „locui" mai mulţi profesori (deşi, uneori, lucrul acesta îi nemulţumeşte profund):

  Birou —/→ Profesor

- Fiecare birou este arondat unei singure catedre (Economie politică, Contabilitate etc.):

  (3) Birou ⟶ Catedră

- Fiecare birou are un număr de telefon unic:

  (4) Birou ⟶ Telefon

Din (3) ar reieşi că (2) este DF tranzitivă, iar relaţia ar trebui descompusă numai în R1 {Birou, Catedră, Telefon} şi R2 {Profesor, Birou}.

Apare însă o situaţie care aruncă în aer algoritmul nostru: *un profesor are un birou alocat altei catedre din care face parte*! Situaţia este reală. Catedrele pot „împrumuta", temporar sau definitiv, birouri sau câte un loc-două din anumite birouri. Pentru profesorii care ocupă un birou alocat altei catedre, schema bazei alcătuite din R1 şi R2 furnizează eronat răspunsul la întrebarea: *Din care catedră face parte fiecare profesor ?*

O soluţie ar fi să se delimiteze, semantic, catedrele care deţin birouri şi catedrele la care sunt afiliaţi profesorii:

R {Profesor, CatedrăProfesor, Birou, CatedrăBirou, Telefon}:

Acum, dependenţele sunt:

(1) Profesor ⟶ Birou

(2) Profesor ⟶ CatedrăProfesor

(3) `Birou` ⟶ `CatedrăBirou`

(4) `Birou` ⟶ `Telefon`

Noua schemă ar fi:

R1 {`Birou`, `CatedrăBirou`, `Telefon`}
şi
R2 {`Profesor`, `CatedrăProfesor`, `Birou`}.

Se poate obiecta că ambele atribute, `CatedrăBirou` şi `CatedrăProfesor`, se referă la aceeaşi entitate, deci seamănă a redundanţă. Cu toate acestea, soluţia elimină problema pierderii de informaţii, deci poate fi însuşită la elaborarea schemei bazei de date.

5.5.2. *Dependenţe tranzitive necanonice*

În exemplele de până acum am avut de lucrat cu dependenţe funcţionale în formă canonică – cele în care destinaţia este alcătuită dintr-un singur atribut. Lucrurile nu sunt grozav de complicate, identificarea dependenţelor fiind o operaţiune lesnicioasă. Din păcate, în unele situaţii este posibil ca tranzitivitatea să se stabilească prin dependenţe ale căror destinaţii sunt compuse. La normalizarea relaţiilor este recomandabil ca, în situaţia unor dependenţe funcţionale de genul: `A` ⟶ `B`, `A` ⟶ `C`, `A` ⟶ `D`, `A` ⟶ `E`, dependenţe ce derivă din rolul de cheie primară pe care îl îndeplineşte atributul `A`, să se verifice dacă nu cumva există:

- fie o dependenţă funcţională simplă, de genul `B` ⟶ `D`;
- fie o dependenţă funcţională cu sursa compusă, de genul (`B`, `C`) ⟶ `D`.

Dacă una dintre cele două afirmaţii de mai sus este adevărată, atunci dependenţa funcţională `A` ⟶ `E` nu este directă.

Cazul nr. 1 – baza de date COMENZI

Luăm în discuţie tabela COMENZI, prezentată în figura 5.14, tabelă care gestionează comenzile pe care firma noastră le-a trimis furnizorilor (în care solicităm materialele/produsele/serviciile de care avem nevoie). Pe baza unei comenzi, un furnizor trimite produsele solicitate, întocmind o factură din care ne va fi remis un exemplar.

COMENZI

NrComanda	NrFactura	CodFurnizor	DataFactura	Valoare	TVADeduct
1501	12344	12	20.06.2004	119000000	19000000
1502	11987	5	15.05.2004	67830000	10830000
1503	45963	2	14.05.2004	71400000	11400000
1504	12456	12	02.07.2004	67592000	10792000
1505	12344	5	27.06.2004	106691830	17034830
1505	12344	12	20.06.2004	101692640	16236640
1507	47890	2	29.05.2004	93886835	14990335

Figura 5.14. Relaţia COMENZI

Stabilim următoarele restricţii :

- o comandă are un număr unic, stabilit de întreprinderea noastră, fiind întocmită pentru un singur furnizor ;
- pentru o comandă, furnizorul va întocmi o singură factură ;
- în schimb, o factură primită de la furnizar poate onora una, două sau mai multe comenzi.

Astfel, atributul `NrComanda` este cheia primară. Ca urmare, se poate scrie :

(1) `NrComanda` ⟶ `NrFactură`
(2) `NrComanda` ⟶ `CodFurnizor`
(3) `NrComanda` ⟶ `DataFactură`
(4) `NrComanda` ⟶ `Valoare`
(5) `NrComanda` ⟶ `TVADeduct`

Facturile sunt întocmite de furnizori (care le numerotează independent între ei) :

(6) (`NrFactură`, `CodFurnizor`) ⟶ `DataFactură`
(7) (`NrFactură`, `CodFurnizor`) ⟶ `Valoare`
(8) (`NrFactură`, `CodFurnizor`) ⟶ `TVADeduct`

Întrucât o factură primită poate onora mai multe comenzi :

(`NrFactură`, `CodFurnizor`) —/→ `NrComanda`

În virtutea ultimelor trei dependenţe funcţionale, rezultă că dependenţele (3), (4) şi (5) sunt tranzitive,

`NrComanda` ⟶ (`NrFactură`, `CodFurnizor`) ⟶ `DataFactură`
`NrComanda` ⟶ (`NrFactură`, `CodFurnizor`) ⟶ `Valoare`
`NrComanda` ⟶ (`NrFactură`, `CodFurnizor`) ⟶ `TVADeduct`

după cum se observă în graful DF din figura 5.15.

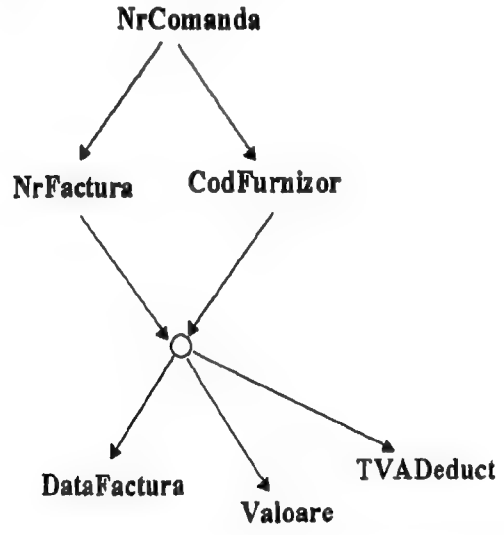


Figura 5.15. DF tranzitive necanonice

Pe baza tranzitivităţii, tabela COMENZI se poate sparge în două relaţii FACTURI_PRIMITE şi COMEZI_FACTURI, ca în figura 5.16.

FACTURI_PRIMITE

NrFactura	CodFurnizor	DataFactura	Valoare	TVADeduct
12344	12	20.06.2004	119000000	19000000
11987	5	15.06.2004	67830000	10830000
45963	2	14.06.2004	71400000	11400000
12456	12	02.07.2004	67592000	10792000
12344	5	27.06.2004	106691830	17034830
47890	2	29.06.2004	93886835	14990335

COMENZI_FACTURI

NrComanda	NrFactura	CodFurnizor
1501	12344	12
1502	11987	5
1503	45963	2
1504	12456	12
1505	12344	5
1506	12344	12
1507	47890	2

Figura 5.16. Eliminarea DF tranzitive din tabela COMENZI

La drept vorbind, nu trebuie un talent deosebit de cârcotaş pentru a recunoaşte că economia de spaţiu şi diminuarea redundanţei sunt mai degrabă simbolice, ba chiar lucrurile se complică prin introducerea unei restricţii referenţiale şi, normal, prin apariţia de două ori a atributelor de legătură `NrFactură` şi `CodFurnizor`.

Există totuşi un merit indiscutabil al noilor relaţii. În relaţia COMENZI nu se poate prelua nici o factură fără a-i cunoaşte comanda (eterna problemă a restricţiei de entitate – nici un atribut component al cheii nu poate avea valori nule). Practica economico-financiară de la noi, dar şi din alte părţi, cunoaşte suficiente cazuri în care firmele primesc facturi direct, fără a înainta o comandă prealabilă. Utilizând proaspetele tabele, FACTURI_PRIMITE şi COMENZI_FACTURI, o factură fără comandă va apărea numai în prima dintre cele două relaţii. Deşi, la prima vedere, am adăugat, şi nu eliminat (aşa cum, generos, sună unul dintre obiectivele normalizării) redundanţă, noua structură este mai „sănătoasă" din punct de vedere relaţional.

Practic, avantajul celor două relaţii din figura 5.15 este cu atât mai evident cu cât numărul facturilor ce onorează două sau mai multe comenzi este mai mare.

Cazul nr. 2 – baza de date EXAMENE

Pentru un plus de credibilitate, să reluăm cazul bazei de date destinate evidenţierii rezultatelor obţinute de studenţi la examene programate în sesiunile de pe parcursul unui an universitar (STUDENŢI_EXAMENE), căreia îi extindem structura:

R {`Matricol, NumePren, An, Modul, Spec, Grupa, CodDisc, DenDisc, NrCredDisc, CodProf, NumeProf, GradProf, DataEx, SalaEx, Nota`},

al cărei dicţionar simplicat este prezentat în tabelul 5.1.

Pentru simplificare, am luat în calcul numai studenţii de la profilul Economic, lungă durată (4-5 ani), cursuri de zi. O linie din R reprezintă rezultatul obţinut de un student la o disciplină într-o sesiune de examinare. Spre exemplu, un student (matricol *EL001103*) poate obţine la disciplina Microeconomie (cod *AE1001*) nota 4 (scuze) în prima sesiune de examinare – ianuarie 2005 (data examenului: *19 ianuarie 2005*), şi nota 9 în a doua sesiune de examinare – februarie 2005 (data examenului: *14/02/2005*). Cei mai puţin dispuşi la efort se pot califica în fazele superioare ale competiţiei (sesiunea mai 2005 ş.a.m.d.).

Tabelul 5.1. Dicţionarul datelor pentru baza de date EXAMENE

Atribut	Descriere	Tip
Matricol	O combinaţie de litere şi cifre ce identifică în mod unic un student	CHAR
NumePren	Numele şi prenumele studentului	CHAR
An	Anul de studiu	NUMERIC
Modul	Modulul de studiu: este o literă ce desemnează un grup de specializări	CHAR
Spec	Specializarea („C" de la contabilitate, „I" de la informatică economică etc.)	CHAR
Grupa	Grupa de studiu	NUMERIC
CodDisc	Codul disciplinei din planul de învăţământ al specializării	CHAR
DenDisc	Denumirea disciplinei	CHAR
NrCredDisc	Numărul de credite alocat disciplinei	NUMERIC
CodProf	Codul profesorului	CHAR
NumeProf	Numele profesorului	CHAR
GradProf	P – profesor, C – conferenţiar, L – lector, A – asistent, R – preparator, A – din afară (colaborator din afara universităţii)	CHAR
DataEx	Data examenului	DATE
SalaEx	Sala de desfăşurare a examenului	CHAR
Nota	Nota obţinută	NUMERIC

Acestea fiind spuse, cheia primară a relaţiei R este, ca şi în cazul STUDENŢI_EXAMENE, (`Matricol`, `CodDisc`, `DataEx`), pe baza căreia pot fi scrise următoarele DF :

(1) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `NumePren`
(2) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `An`
(3) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `Modul`
(4) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `Spec`
(5) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `Grupa`
(6) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `DenDisc`
(7) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `NrCredDisc`
(8) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `CodProf`
(9) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `NumeProf`
(10) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `GradProf`
(11) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `SalaEx`
(12) (`Matricol`, `CodDisc`, `DataEx`) ⟶ `Nota`

Relaţia R este în prima formă normalizată. După cum am văzut în capitolul precedent, pentru a demonstra că nu este în 2NF trebuie să găsim în ansamblul DF (1)-(12) măcar o DF parţială. Lucru destul de simplu, dacă ţinem seama de următoarele DF, în care sursa o constituie unul dintre atributele componente ale cheii.

- Matricolul identifică în mod unic un student înscris la o specializare într-un an de studii :

  (13) `Matricol` ⟶ `NumePren`
  (14) `Matricol` ⟶ `An`
  (15) `Matricol` ⟶ `Modul`

(16) Matricol ⟶ Spec

(17) Matricol ⟶ Grupa

- CodDisc identifică în mod unic fiecare discplină din planul de învăţământ al specializării:

  (18) CodDisc ⟶ DenDisc

  (19) CodDisc ⟶ NrCredDisc

Pe baza ultimelor şapte DF, relaţia universală R se poate descompune astfel:

R1 {Matricol, NumePren, An, Modul, Spec, Grupa}

R2 { CodDisc, DenDisc, NrCredDisc}

R3 {Matricol, CodDisc, CodProf, NumeProf, GradProf, DataEx, SalaEx, Nota}

Cele trei relaţii sunt în 2NF, iar până aici similitudinea cu discuţia pricinuită de relaţia STUDENŢI_EXAMENE este tulburătoare. Aducerea BD în 3NF echivalează cu eliminarea dependenţelor tranzitive din R1, R2 şi R3. În R1, singurele DF existente sunt (13)-(17), deci această relaţie este în 3NF. Nici R2 nu conţine dependenţe tranzitive; în schimb, R3, da:

- CodProf identifică în mod unic un profesor:

  (20) CodProf ⟶ NumeProf

  (21) CodProf ⟶ GradProf

Pe baza DF (20) şi (21), putem spune că (9) şi (10) sunt tranzitive, aşa încât rupem R3 în R31 {CodProf, NumeProf, GradProf} şi R32 {Matricol, CodDisc, CodProf, DataEx, SalaEx, Nota}. Astfel, în 3NF, relaţia universală R are o schemă alcătuită din patru relaţii, după cum urmează:

STUDENŢI {Matricol, NumePren, An, Modul, Spec, Grupa}

DISCIPLINE { CodDisc, DenDisc, NrCredDisc}

PROFESORI {CodProf, NumeProf, GradProf}

NOTE {Matricol, CodDisc, CodProf, DataEx, SalaEx, Nota}

Nici una dintre cele patru relaţii nu conţine DF tranzitive. Cu toate acestea, schema este departe de a fi optimă. Faţă de DF discutate, mai există două restricţii neluate în calcul la normalizare:

- La o serie de curs (seriile de curs se constituie, pentru fiecare disciplină, la nivel de an, modul, specializare), titular este un singur profesor:

  (22) (CodDisc, An, Modul, Spec) ⟶ CodProf

- Studenţii unei specializări susţin examenul la o disciplină în aceeaşi sală (şi aceeaşi zi):

  (23) (An, Modul, Spec, DataEx) ⟶ SalaEx

Să recapitulăm: în relaţia NOTE există DF:

(8) (Matricol, CodDisc, DataEx) ⟶ CodProf

dar şi

(22) (CodDisc, An, Modul, Spec) ⟶ CodProf

Deoarece există DF:

(24) Matricol ⟶ An, Modul, Spec

rezultă că (8) este o DF necanonică, aşa încât relaţia NOTE1 s-ar descompune în:

DISC_SPEC_PROFI {CodDisc, An, Modul, Spec, CodProf}

şi

NOTE2 {Matricol, CodDisc, DataEx, SalaEx, Nota}

Câştigul este considerabil. Analog se poate proceda cu DF (23), discuţia transferându-se în relaţia NOTE2, în care, din calitatatea de cheie primară a celor trei atribute, există dependenţa:

(11) (Matricol, CodDisc, DataEx) ⟶ SalaEx

dar şi DF:

(23) (An, Modul, Spec, CodDisc, DataEx) ⟶ SalaEx

Pe baza dependenţei (24), relaţia NOTE2 se sparge în SĂLI_EX {An, Modul, Spec, CodDisc, DataEx, SalaEx} şi NOTE {Matricol, CodDisc, DataEx, Nota}.

Ajungem, astfel, la o cu totul altă structură a bazei de date, mult mai bună decât precedenta:

STUDENTI {Matricol, NumePren, An, Modul, Spec, Grupa}
DISCIPLINE {CodDisc, DenDisc, NrCredDisc}
PROFESORI {CodProf, NumeProf, GradProf}
DISC_SPEC_PROFI {CodDisc, An, Modul, Spec, CodProf}
SALI_EX {An, Modul, Spec, CodDisc, DataEx, SalaEx}
NOTE {Matricol, CodDisc, DataEx, Nota}

Ca de obicei, dependenţele tranzitive, canonice sau necanonice se observă mult mai lejer pe graful DF - vezi figura 5.17. Este suficient să decupăm câte o relaţie pentru fiecare sursă simplă şi compusă din graf, sursă ce devine automat cheie a relaţiei respective, şi vom obţine structura de mai sus.

Structura finală îndeplineşte şi cerinţele formei normale Boyce-Codd, ca fiecare sursă de DF să fie cheie candidată. Putem institui şi o regulă: atunci când descompunerea se face urmând filiera clasică:

- stabilirea cheii primare a relaţiei iniţiale (universale), apoi
- eliminarea, dintre dependenţele funcţionale ce decurg din calitatea de cheie primară a relaţiei iniţiale, a celor parţiale, apoi
- eliminarea, din toate relaţiile obţinute în pasul anterior, a tuturor dependenţelor tranzitive (în care surse sunt cheile primare ale respectivelor relaţii), trebuie verificat dacă în relaţia iniţială mai existau dependenţe cu sursa compusă ce nu apar drept chei primare sau candidate în nici una dintre relaţiile finale obţinute.

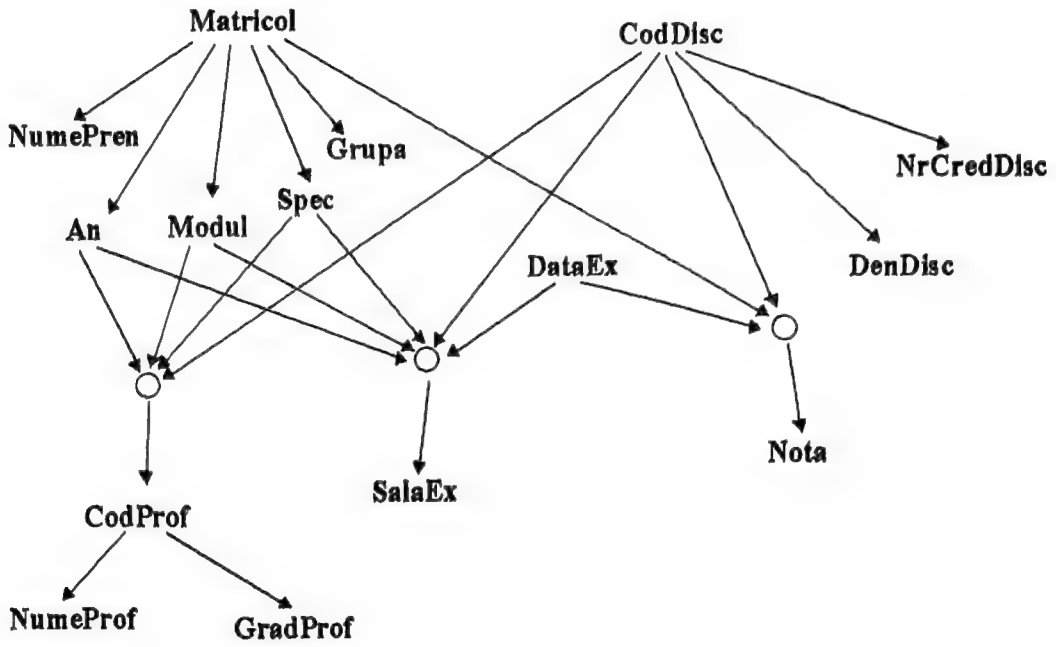


Figura 5.17. Graful DF pentru baza de date EXAMENE

5.5.3. *Relaţii semantice imposibil de preluat în dependenţe funcţionale*

Un alt aspect delicat al normalizării ţine de dificultatea preluării în schemă, sub formă de dependenţe funcţionale, a unor relaţii semantice. Pentru o primă ilustrare să luăm relaţia PROFI_DISCIPLINE din figura 5.18, ce conţine date legate de profesori şi cursurile predate într-o facultate.

PROFI_DISCIPLINE

Id Prof	Nume Prof	Catedră	Cod Disc	DenDisc	LbStr Prof	LbStr Disc
111	Airinei Dinu	Informatică Economică	AI1101	Introducere în informatica economică	FR	FR
112	Fotache Marin	Informatică Economică	AI1101	Introducere în informatica economică	EN	EN
112	Fotache Marin	Informatică Economică	AI1101	Introducere în informatica economică	FR	FR

Figura 5.18. Cursuri predate în limbi străine

Un profesor este identificat prin `IdProf` şi face parte dintr-o singură catedră:

(1) `IdProf` ⟶ `NumeProf`

(2) `IdProf` ⟶ `Catedră`

Un curs este identificat prin cod şi arondat unei singure catedre:

(3) `CodDisc` ⟶ `DenDisc`

(4) `CodDisc` ⟶ `Catedră`

În schimb:

- un profesor predă mai multe discipline:
  `IdProf` —/→ `CodDisc`
- o disciplină poate fi predată de mai mulţi profesori:
  `CodDisc` —/→ `IdProf`
- un profesor poate preda în mai multe limbi străine:
  `IdProf` —/→ `LbStrProf`
- un curs poate fi predat în mai multe limbi străine:
  `CodDisc` —/→ `LbStrDisc`
- un profesor poate preda acelaşi curs în mai multe limbi străine:
  (`IdProf`, `CodDisc`) —/→ `LbStrDisc`
- un profesor poate preda în aceeaşi limbă străină două sau mai multe cursuri:
  (`IdProf`, `LbStrDisc`) —/→ `CodDisc` şi
  (`IdProf`, `LbStrProf`) —/→ `CodDisc`

Practic, graful DF se prezintă ca în figura 5.19.

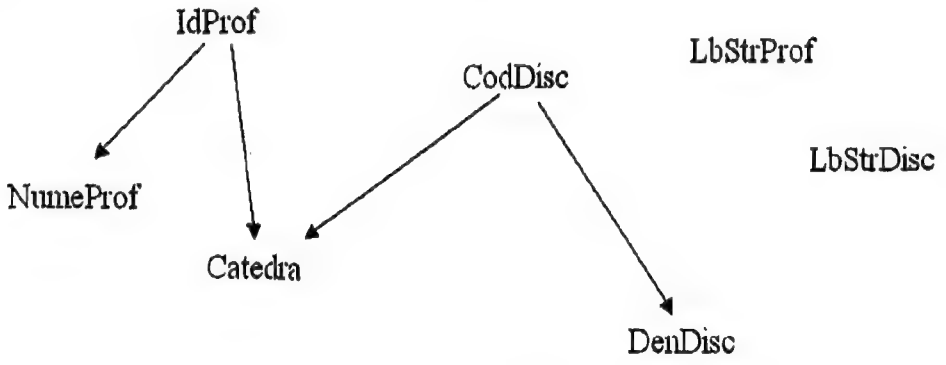


Figura 5.19. Graful DF pentru relaţia PROFI_DISCIPLINE

Nemulţumirea legată de informaţia reprezentată prin graf ţine de faptul că un profesor ştie anumite limbi străine, iar o disciplină este predată în câteva limbi, nicidecum în toate. Cu alte cuvinte, schema de mai sus ridică probleme la stocarea informaţiilor: *care sunt limbile străine cunoscute de fiecare profesor* şi *care sunt limbile străine în care se predă fiecare disciplină*.

Atributele `LbStrProf` şi `LbStrDisc` rămân suspendate în graf. Ce-i de făcut? O idee, nu întotdeauna rezonabilă, este de a le plasa într-o relaţie împreună cu celelalte „rădăcini" ale grafului, obţinându-se astfel schema:

PROFI {`IdProf`, `NumeProf`, `Catedra`},

DISCIPLINE {`CodDisc`, `DenDisc`, `Catedra`} şi

PROFI_DISC_LBSTR {`IdProf`, `CodDisc`, `LbStrProf`, `LbStrDisc`}

Predicatul care ar exprima oricare tuplu al ultimei relaţii ar fi: un profesor (`IdProf`) care cunoaşte o limbă străină (`LbStrProf`) predă o disciplină (`CodDisc`)

într-una dintre limbile străine pe care le vorbeşte (LbStrDisc). Firește, apare întrebarea: Ce relaţie trebuie să existe între valorile LbStrProf şi LbStrDisc într-un tuplu, ştiind că disciplina este predată şi în alte limbi decât LbStrProf, iar profesorul ştie şi alte limbi străine decât LbStrDisc?

O soluţie acceptabilă în multe cazuri constă în introducerea unor atribute suplimentare, tocmai pentru a „fixa" sub formă de dependenţe restricţiile respective. Spre exemplu, pentru a „formaliza" restricţia, *un profesor ştie anumite limbi străine*, se poate introduce un atribut, Nivel, pentru a ilustra gradul de comunicare al profesorului în limba străină respectivă:

(IdProf, LbStrProf) ⟶ Nivel

De asemenea, pentru a indica în ce limbi străine este disponibil un curs, se introduce atributul SerieCurs. La un obiect, o serie de curs se constituie pentru o singură limbă străină şi cade în responsabilitatea unui singur profesor:

(CodDisc, SerieCurs) ⟶ LbStrDisc

(CodDisc, SerieCurs) ⟶ IdProf

Graful obţinut nu mai conţine atribute „suspendate" - vezi figura 5.20 -, iar schema desprinsă din acesta este mult mai bună:

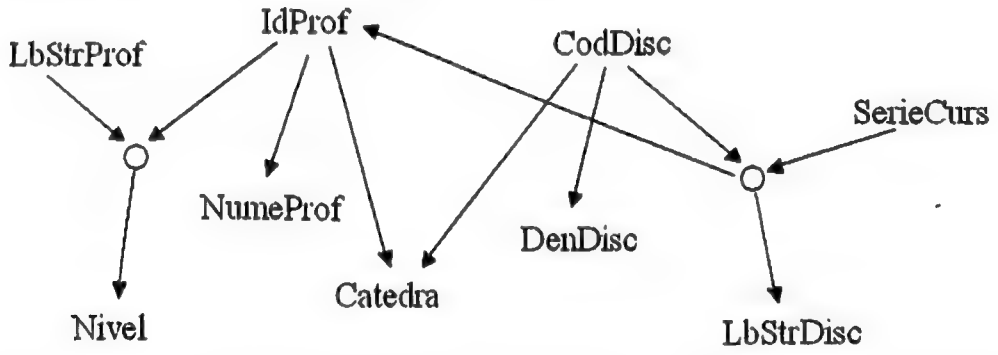


Figura 5.20. Noul graf DF pentru relaţia PROFI_DISCIPLINE ameliorată

PROFI {IdProf, NumeProf, Catedra},
DISCIPLINE {CodDisc, DenDisc, Catedra},
PROFI_LBSTR {IdProf, LbStrProf, Nivel} şi
SERII_CURS_LBSTR {CodDisc, SerieCurs, LbStrDisc, IdProf}.

Deşi oarecum artificială, tehnica introducerii de atribute suplimentare, care nu sunt evidente în momentul construirii relaţiei universale, este deseori salutară, fiind folosită pe scară largă de practicieni. În plus, putem vorbi de un argument suplimentar în favoarea ideii de ciclicitate a normalizării, de reluare a o serie de paşi până se obţine o schemă considerată mulţumitoare.

Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 se întoarce!

Revenim la un caz „clasat" pentru 3NF, cel al relaţiei BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3. Combinaţia care diferenţiază orice linie de toate celelalte este

(`Cota, Autor, CuvîntCheie`). Intuim însă un aspect pe care dependenţele şi, implicit, normalizarea îl scapă, şi anume că autorii şi cuvintele-cheie se referă la o carte (adică `ISBN`), şi nu la un exemplar fizic din depozit (`Cota`). Deci graful DF nu ar trebui să arate ca în figura 5.3, ci precum în figura 5.21. Din păcate, acesta este un aspect pe care îl intuim, dar e mai greu de demonstrat „ştiinţific" prin normalizare. Un argument în plus pentru a nu considera normalizarea drept un proces infailibil, ci doar un ghid în construirea unei scheme a BDR cât mai bune.

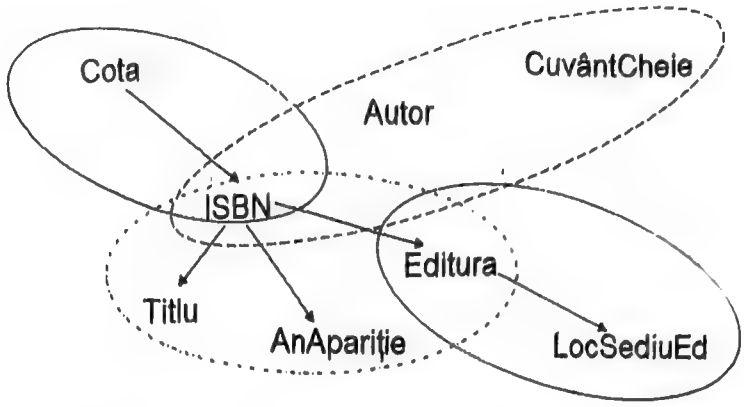


Figura 5.21. Autorii şi cuvintele-cheie corespund titlurilor, nu exemplarelor din cărţi

Pe baza grafului de mai sus, din schema bazei de date în 3NF se schimbă ultima tabelă, TITLURI_AUTORI_CUVINTECHEIE2 {`ISBN, Autor, CuvântCheie`}.

Conţinutul celor patru relaţii este acum cel din figura 5.22.

Se observă cu ochiul liber o diminuare a volumului ultimei tabele, astfel încât ne-am încumetat să reprezentăm toate liniile. Rămâne, în schimb, un grad de redundanţă chiar şi în forma „comprimată" a tabelei.

Similar relaţiei PROFI_DISCIPLINE din figura 5.18, putem încerca să adăugăm în BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 câteva atribute care să atragă în dependenţe atributele `Autor` şi `CuvântCheie`. Astfel, pornim de la faptul că, pe coperta unei cărţi, autorii sunt trecuţi într-o anumită ordine, deloc întâmplătoare, iar la extragerea informaţiilor despre cărţi această ordine trebuie respectată. Prin urmare, folosim atributul `OrdineCopertă`, iar DF este:

(`ISBN, OrdineCopertă`) ⟶ `Autor`

De asemenea, pentru a putea extrage cărţile legate de unul sau mai multe cuvinte-cheie în ordinea relevanţei, introducem atributul `Relevanţă`, ce indică nivelul la care este tratată într-o carte o anumită noţiune/sintagmă (cuvânt-cheie):

(`ISBN, CuvântCheie`) ⟶ `Relevanţă`

Pe baza acestor două noi dependenţe, graful din figura 5.21 se modifică după cum este sugerat în figura 5.23.

Din noul graf, relaţiile construite sunt aproape impecabile:

EDITURI {`Editura, LocSediuEd`}

CĂRŢI {`ISBN, Titlu, AnApariţie, Editură`}

EDITURI

Editura	LocSediuEd
Polirom	Iaşi

TITLURI

ISBN	Titlu	Editura	AnAparitie
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Polirom	2002
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Polirom	2001

EXEMPLARE

Cotă	ISBN
III-13421	973-683-889-7
III-13422	973-683-889-7
III-13423	973-683-889-7
III-10678	973-683-709-2
III-10679	973-683-709-2

TITLURI_AUTORI_CUVINTECHEIE2

ISBN	Autor	CuvântCheie
973-683-889-7	Marin Fotache	baze de date
973-683-889-7	Marin Fotache	SQL
973-683-889-7	Marin Fotache	proceduri stocate
973-683-889-7	Marin Fotache	FoxPro
973-683-889-7	Marin Fotache	formulare
973-683-889-7	Marin Fotache	orientare pe obiect
973-683-889-7	Marin Fotache	client-server
973-683-889-7	Marin Fotache	Web
973-683-889-7	Ioan Brava	baze de date
973-683-889-7	Ioan Brava	SQL
973-683-889-7	Ioan Brava	proceduri stocate
973-683-889-7	Ioan Brava	FoxPro
973-683-889-7	Ioan Brava	formulare
973-683-889-7	Ioan Brava	orientare pe obiect
973-683-889-7	Ioan Brava	client-server
973-683-889-7	Ioan Brava	Web
973-683-889-7	Cătălin Strâmbei	baze de date
973-683-889-7	Cătălin Strâmbei	SQL
973-683-889-7	Cătălin Strâmbei	proceduri stocate
973-683-889-7	Cătălin Strâmbei	FoxPro
973-683-889-7	Cătălin Strâmbei	formulare
973-683-889-7	Cătălin Strâmbei	orientare pe obiect
973-683-889-7	Cătălin Strâmbei	client-server
973-683-889-7	Cătălin Strâmbei	Web
973-683-889-7	Liviu Creţu	baze de date
973-683-889-7	Liviu Creţu	SQL
973-683-889-7	Liviu Creţu	proceduri stocate
973-683-889-7	Liviu Creţu	FoxPro
973-683-889-7	Liviu Creţu	formulare
973-683-889-7	Liviu Creţu	orientare pe obiect
973-683-889-7	Liviu Creţu	client-server
973-683-889-7	Liviu Creţu	Web
973-683-709-2	Marin Fotache	baze de date
973-683-709-2	Marin Fotache	algebră relaţională
973-683-709-2	Marin Fotache	SQL

Figura 5.22. O 3NF mai bună pentru relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3

AUTORI_CĂRŢI {ISBN, OrdineCopertă, Autor}
CĂRŢI_CUVINTE_CHEIE {ISBN, CuvântCheie, Relevanţă}
EXEMPLARE_CĂRŢI {Cota, ISBN}

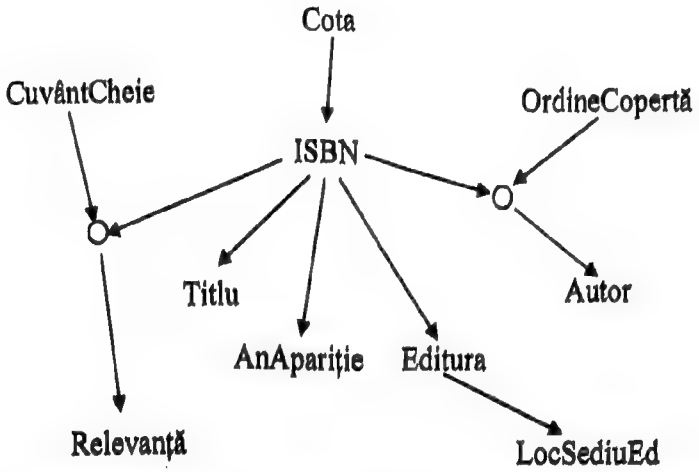


Figura 5.23. Noul graf pentru relaţia BIBLIOTECĂ (cu două atribute noi)

5.5.4. *Simetrie şi temporalitate în dependenţele funcţionale*

Revenim la relaţia COMENZI, descrisă în paragraful 5.5.2 (figura 5.14), pentru care schimbăm condiţia de derulare a operaţiunilor. Astfel, dacă în paragraful 5.5.2 am inventariat dependenţele în situaţia în care o comandă este onorată printr-o singură factură, dar o factură poate să corespundă mai multor comenzi, discutăm în cele ce urmează relaţia COMENZI2 pentru care, după recepţionarea unei comenzi primite de la firma noastră, furnizorul o „rezolvă" expediind produsele comandate într-o singură livrare şi, prin urmare, întocmeşte o singură factură. În plus, o factură nu poate onora decât maximum o comandă. Graful DF ia forma din figura 5.24.

COMENZI2
- *O comandă este onorată de o singură factură*
- *O factură onorează o singură comandă*

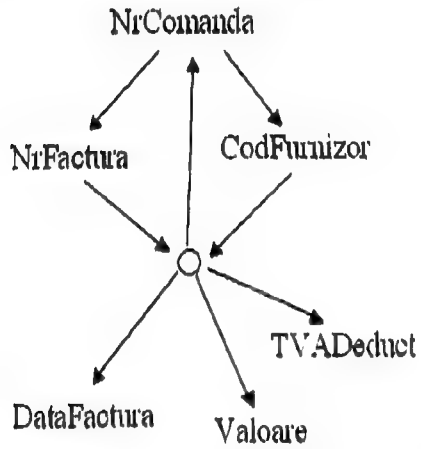


Figura 5.24. O (altă) dependenţă simetrică

De data aceasta, graful nu mai este aşa de prompt în furnizarea schemei bazei de date în BCNF. Dacă am urma regula *o sursă = o relaţie*, atunci COMENZI2 s-ar descompune în:

COMENZI21 {NrComandă, NrFactură, CodFurnizor}
şi
FACTURI21 {NrFactură, CodFurnizor, DataFactură, Valoare, TVADeduct, NrComandă}.

Redundanţa este însă evidentă, deoarece graful conţine două dependenţe simetrice, NrComandă ⟶ (NrFactură, CodFurnizor) şi (NrFactură, CodFurnizor) ⟶ NrComandă. Pe care o eliminăm? Pare mai logic pe prima, deoarece sursa (NrFactură, CodFurnizor) are şi alte destinaţii. Practic, relaţia COMENZI2 păstrează aceeaşi structură în 2NF, 3NF şi BCNF:

COMENZI2 {NrFactură, CodFurnizor, DataFactură, Valoare, TVADeduct, NrComandă}

În paragraful 4.1.4 recomandam ca, atunci când pentru identificarea unui/unei obiect/ operaţiuni/tranzacţii există mai multe atribute/combinaţii, prin urmare, mai multe chei candidate, la reprezentarea dependenţelor funcţionale să se păstreze numai cele în care sursa este un identificator (cheie primară). Cele în care sursele sunt chei alternative, iar destinaţiile, identice cu ale identificatorului primar trebuie eliminate. Altfel spus, dacă:

X ⟶ A, X ⟶ B, X ⟶ C, Y ⟶ A, Y ⟶ B, Y ⟶ C şi X ⟶ Y, dar şi Y ⟶ X,

atunci se alege X (sau Y) drept identificator primar şi se păstrează numai dependenţele:

X ⟶ A, X ⟶ B, X ⟶ C, X ⟶ Y.

Prelungim discuţia de la relaţia COMENZI2 introducând cu bună ştiinţă atributul DataComandă pentru a şti în ce zi a fost întocmită fiecare comandă trimisă furnizorului. Altfel scris, relaţia conţine atributele COMENZI3 {NrComanda, DataComanda, NrFactura, CodFurnizor, DataFactura, Valoare, TVADeduct}, iar raportul comenzi-facturi este de tip unu-la-unu. În postura „candidaţilor" X şi Y avem atributul NrComandă, respectiv combinaţia (NrFactura, CodFurnizor). Seria de DF care se înregistrează în COMENZI3 este:

NrComanda ⟶ DataComanda	(NrFactura, CodFurnizor) ⟶ NrComanda
NrComanda ⟶ NrFactura	(NrFactura, CodFurnizor) ⟶ DataComanda
NrComanda ⟶ CodFurnizor	(NrFactura, CodFurnizor) ⟶ DataFactura
NrComanda ⟶ DataFactura	(NrFactura, CodFurnizor) ⟶ Valoare
NrComanda ⟶ Valoare	(NrFactura, CodFurnizor) ⟶ TVADeduct
NrComanda ⟶ TVADeduct	

Dacă dintre cei doi candidaţi s-ar alege NrComandă, atunci graful ar arăta precum varianta 1 din figura 5.25, în timp la alegerea combinaţiei (NrFactura, CodFurnizor) forma finală a grafului ar fi cea din varianta 2.

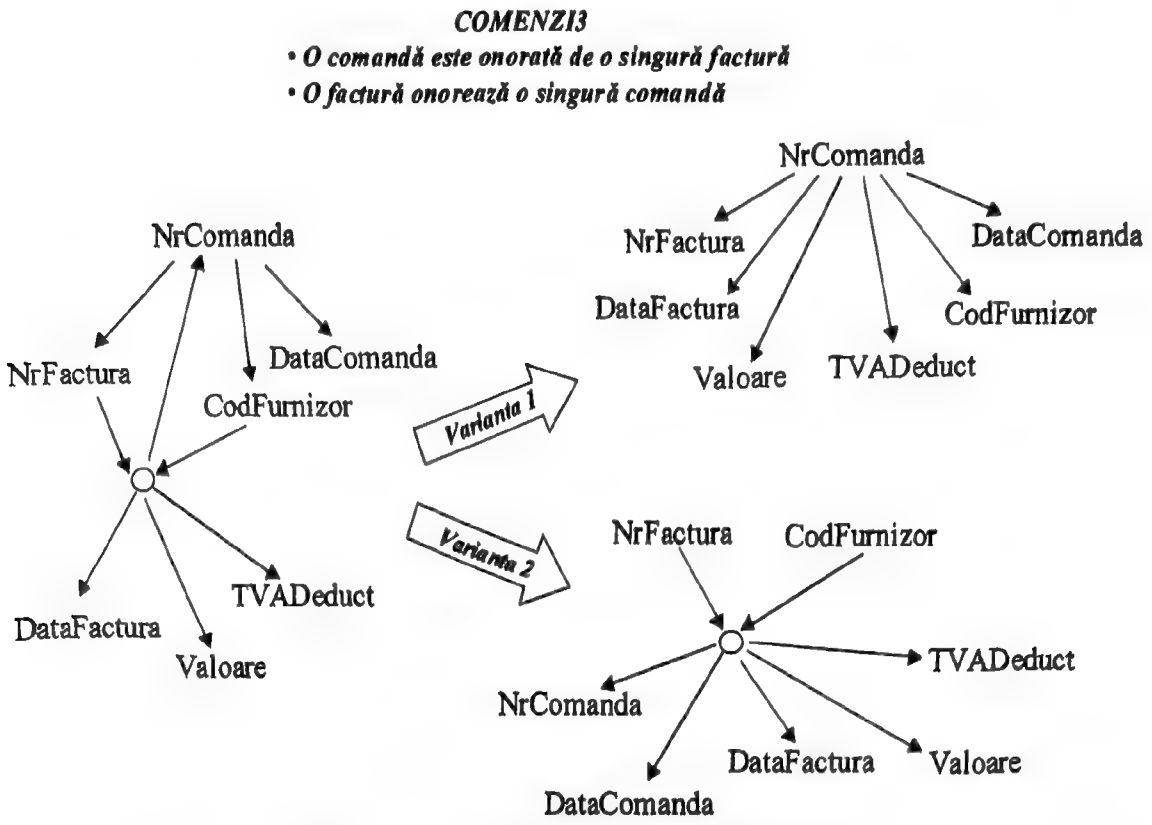


Figura 5.25. O (altă) dependenţă simetrică

În ambele variante, relaţiile obţinute sunt identice, doar că alcătuirea cheii primare este diferită:

COMENZI3_VARIANTA1 {NrComandă, DataComanda, NrFactură, CodFurnizor, DataFactură, Valoare, TVADeduct}
COMENZI3_VARIANTA2 {NrFactură, CodFurnizor, DataFactură, Valoare, TVADeduct, NrComandă, DataComanda}

Dacă, din punctul de vedere al dependenţelor funcţionale, oricare variantă este corectă, în practică, relaţia COMENZI3_VARIANTA2 nu este dezirabilă, întrucât ordinea operaţiunilor este rigidă: mai întâi se redactează comanda şi se transmite furnizorului, iar apoi furnizorul o onorează. Comanda este deci anterioară facturii. Pentru relaţia incriminată, inserarea unei comenzi nu este posibilă decât în momentul onorării ei (altminteri, atributele cheie ar avea valori NULL), iar cunoaşterea comenzilor (încă) neonorate este imposibilă.

Urmarea ar fi că, în situaţia dependenţelor simetrice, eliminarea simetriei trebuie să ţină cont şi de eventuala succesiune a operaţiunilor, privilegiindu-se sursele în ordinea precedenţei lor. Din acest punct de vedere, COMENZI3_VARIANTA1 este relaţia ce pare a rezolva problema. Obiecţia celor ce se opun folosirii valorilor NULL este una îndreptăţită, deoarece, din momentul întocmirii comenzi până la cel al livrării (facturii) corespunzătoare, atributele NrFactură, CodFurnizor, DataFactură, Valoare, TVADeduct sunt NULL.

Toate aceste discuţii apar datorită faptului că dependenţele funcţionale şi, implicit, graful acestora nu oferă întotdeauna informaţii legate de succesiunea operaţiunilor,

aşa încât, cu riscul de a complica graful, putem apela la un artificiu care să sugereze ordinea momentelor la care este valabilă fiecare DF – vezi figura 5.26.

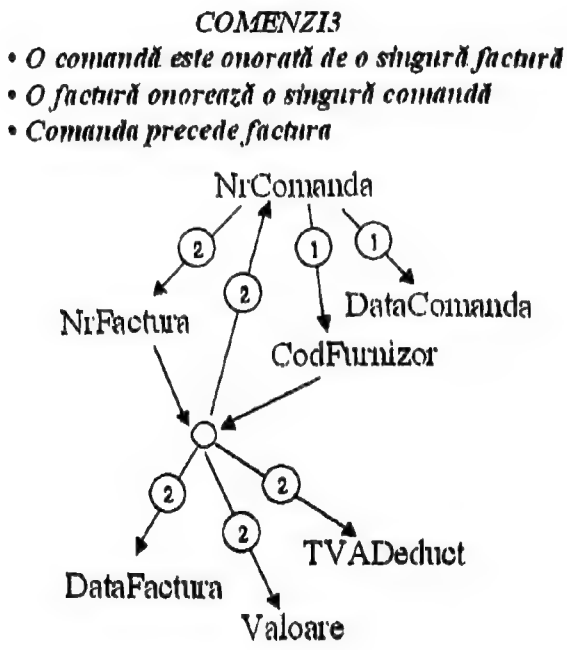


Figura 5.26. Graful din figura 5.24 completat cu elemente care să indice ordinea „producerii" dependenţelor

Astfel, s-ar putea construi o relaţie pentru fiecare sursă de DF şi „moment" consemnat în graf. Prima este asociată momentului întocmirii comenzii:

NUMAI_COMENZI3 {NrComandă, DataComanda, CodFurnizor}

Dintre DF simetrice (NrFactura, CodFurnizor) ⟶ NrComanda şi NrComanda ⟶ NrFactura, respectiv NrComanda ⟶ CodFurnizor, se elimină prima, aşa încât rezultă alte două relaţii:

COMENZI3_FACTURI {NrComandă, NrFactură}
şi
FACTURI3 {NrFactură, CodFurnizor, DataFactură, Valoare, TVADeduct}.

Deşi pare ciudat, iar fragmentarea bazei greu suportabilă, această structură este extrem de flexibilă, iar valorile NULL sunt eliminate în totalitate. Un alt efect colateral benefic este că, acum, se rezolvă şi problema sosirii facturilor înaintea comenzii, sau chiar a facturilor fără comenzi. Practicienii ştiu că aceste situaţii sunt greu de evitat, mai ales în situaţia ţării noastre, în care rigoarea nu reprezintă o vocaţie naţională.

5.5.5. *Grafuri şi BCNF*

Revenim la relaţia COMENZI descrisă în paragraful 5.5.2 (figura 5.14), pentru care schimbăm, încă o dată, condiţiile de derulare a operaţiunilor (COMENZI4). Astfel,

chiar dacă o factură nu poate corespunde decât unei singure comenzi, o comandă poate fi onorată în tranşe, prin mai multe livrări/transporturi, şi, în consecinţă, mai multe facturi – vezi figura 5.27.

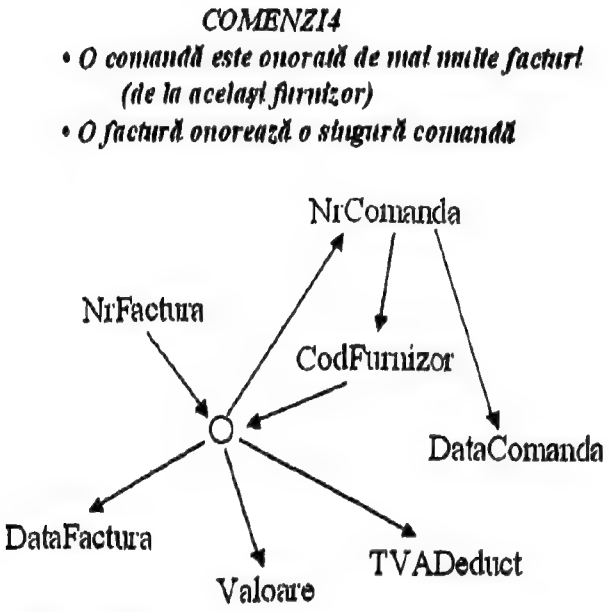


Figura 5.27. Situaţie ce necesită aducerea relaţiei în BCNF

Urmând „preceptele" construirii relaţiilor pe baza grafului, ar trebui să obţinem schema:

COMENZI41 {NrComandă, CodFurnizor, DataComanda}
şi
FACTURI41 {NrFactură, CodFurnizor, DataFactură, Valoare, TVADeduct, NrComandă}.

Ceea ce ne interesează în cea mai mare măsură în graf este, să-i zicem, semi-circularitatea dependenţelor, specifică discuţiilor legate de BCNF. Practic, descompunerea în COMENZI41 Şi FACTURI41 este oarecum incorectă, deoarece schema nu este în BCNF. În COMENZI4 se manifestă atât dependenţa (NrFactură, CodFurnizor) ⟶ NrComandă, cât şi NrComandă ⟶ CodFurnizor. Descompunerea corectă este:

COMENZI_FURNIZORI {NrComandă, CodFurnizor, DataComanda}
şi
FACTURI {NrFactură, NrComandă, DataFactură, Valoare, TVADeduct}.

În concluzie, atunci când relaţiile sunt construite direct din graful DF, trebuie verificat în prealabil dacă, în graf, vreun atribut sau grup de atribute X dintr-o sursă compusă de genul:

(X, Y) ⟶ A,

(X, Y) ⟶ B,

(X, Y) ⟶ C

nu este destinaţie a unei dependenţe de genul A ⟶ X. Dacă da, atunci se păstrează A ⟶ X, iar în celelalte DF, sursa (X, Y) se înlocuieşte cu (A, Y), adică:

(A, Y) ⟶ B,

(A, Y) ⟶ C

Ilustrarea acestui algoritm al transformării grafului din figura 5.27 este prezentată în figura 5.28.

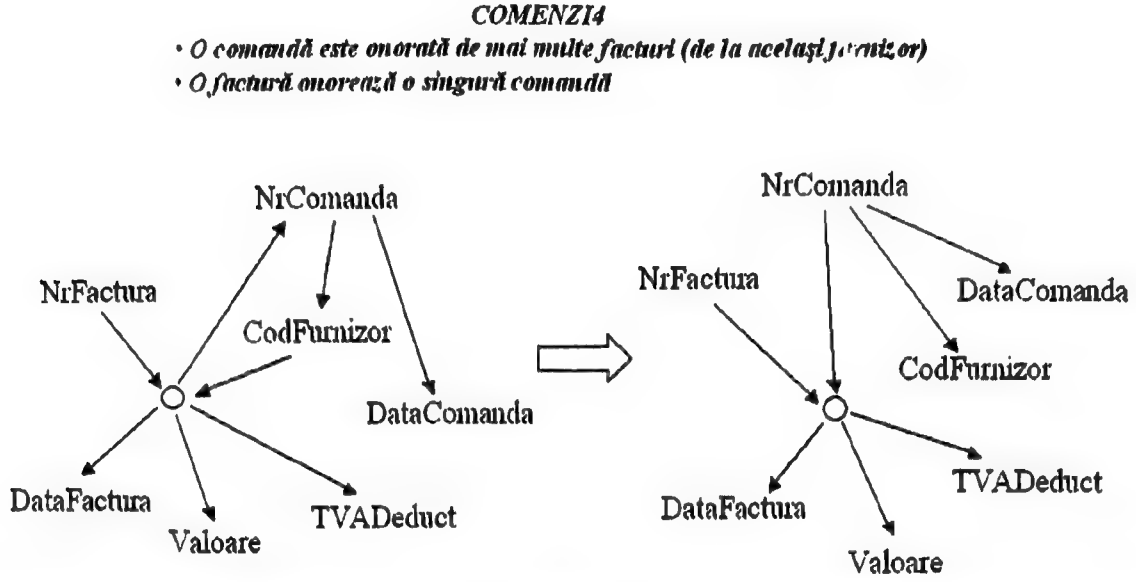


Figura 5.28. Transformarea grafului din figura 5.27 pentru aducerea relaţiei în BCNF

CAPITOLUL 6

Dependenţe multivaloare şi de joncţiune. Formele normalizate 4 şi 5

Formele normalizate 4 şi 5 constituie cea mai dificilă parte din procesul de normalizare a unei baze de date. Aceasta este vestea rea. Vestea bună este că rareori proiectanţii bazei trebuie să ajungă în acest stadiu al discuţiilor, mai ales în ceea ce priveşte 5NF, deoarece majoritatea cazurilor practice pot fi rezolvate onorabil prin aducerea bazei în 3NF/BCNF sau, uneori, în 4NF.

Dependenţele funcţionale discutate până la acest paragraf sunt relativ uşor de identificat şi explicat. În prezentul capitol vor fi prezentate două tipuri de dependenţe mult mai greu de digerat. Este şi motivul pentru care majoritatea lucrărilor dedicate normalizării bazelor de date relaţionale fie le prezintă rapid, fără prea multe explicaţii, păstrând un aer discret, fie nu le prezintă deloc, ceea ce le pune la adăpost de orice eroare.

6.1. Dependenţe multivaloare

Dependenţa *multivaloare* sau *multivalorică* şi, implicit, a patra formă normală au fost introduse la câţiva ani după dependenţele funcţionale şi primele trei forme normale, prin 1977, de către Ronald Fagin[1]. Fie relaţia R $\{A_1, A_2, \ldots, A_n\}$ şi X, Y şi Z trei subansambluri de atribute ale ansamblului $\{A_1, A_2, \ldots, A_n\}$. Există o dependenţă multivaloare (DMV) între X şi Y dacă şi numai dacă:

- la fiecare apariţie (valoare) a lui X poate fi asociată una sau mai multe apariţii (valori) ale lui Y;
- această asociaţie nu depinde de apariţiile lui Z.

Se notează X $\longrightarrow\!\!\!\!\rightarrow$ Y sau X $\longrightarrow\!\!\!\!\rightarrow$ Y | Z şi se spune că atributul Y *este multidependent faţă de atributul* X sau că *atributul* X *multideterminină pe* Y. Deşi Fagin nu preciza acest lucru în lucrarea sa[2], Date este de părere că se poate pune problema

1. [Fagin 77].
2. Similară lui Fagin este şi abordarea din Garcia-Molina, Ullman şi Widom, [Garcia-Molina ş.a. 02], p. 119.

unei DMV numai în condiţiile existenţei a minimum trei atribute[3]. De asemenea, dacă există DMV X ⟶→ Y | Z, atunci obligatoriu şi X ⟶→ Z | Y.

Dependenţa multivaloare este percepută şi prezentată diferit, chiar, pe alocuri, eronat. Pentru Rebecca Riordan, Florescu ş.a., simpla repetare a unui atribut este un indicator cât se poate de clar că avem de-a face cu o DMV, ceea ce nu este prea adevărat. Spre exemplu, R. Riordan ia un model identic celui din figura 6.1[4]. Ceea ce nu precizează autoarea ţine de faptul că există DMV DenProdus ⟶→ UnitateMăsură numai în condiţiile în care *fiecare furnizor vinde toate produsele în toate unităţile de măsură ale produsului respectiv*!!!

R

DenProdus	NumeFurnizor	UnitateMăsură
Produs 1	Furnizor 1	cutii 250 ml
Produs 1	Furnizor 1	cutii 500 ml
Produs 1	Furnizor 1	cutii 1 l
Produs 2	Furnizor 2	cutii 250 ml
Produs 2	Furnizor 2	cutii 500 ml
Produs 2	Furnizor 2	cutii 1 l
Produs 3	Furnizor 3	cutii 250 ml
Produs 3	Furnizor 3	cutii 500 ml
Produs 3	Furnizor 3	cutii 1 l

Figura 6.1. Exemplul lui R. Riordan pentru DMV

În [Florescu ş.a. 99] lucrurile sunt şi mai departe de adevăr: „în cazul în care unui atribut îi corespund mai multe valori ale altui atribut este vorba de o dependenţă multivaloare"[5], speţa prezentată fiind Client ⟶→ NrComandă, pe motiv că unui client îi corespund mai multe comenzi. Orice comentariu este de prisos.

Nici în lucrarea lui Octavian Bâscă DMV nu stă chiar pe roze. Definiţia teoretică merită a fi reţinută, anume că X ⟶→ Y dacă şi numai dacă oricare ar fi tuplurile u şi v pentru care $u[X] = v[X]$, relaţia conţine obligatoriu şi tuplurile t şi s care îndeplinesc condiţiile[6]:

- $t[X] = u[X]$,
- $t[Y] = u[Y]$,
- $t[R - X - Y] = v[R - X - Y]$,
- $s[X] = v[X]$,
- $s[Y] = v[Y]$,
- $s[R - X - Y] = u[R - X - Y]$

Cum aproape întodeauna se verifică postulatul „teroria ca teoria, dar practica ne omoară", exemplul R {Curs, Profesor, Ora, Sala, Student, AnStudii}, în care

3. [Date86], p.384
4. [Riordan 99], p. 40.
5. [Florescu ş.a.], p. 99.
6. [Bâscă 97], p. 100.

Curs ⟶→ (Ora, Sala), este mult mai puţin elocvent, iar eu îl consider incorect chiar în condiţiile restricţiilor formulate.

În schimb, într-una dintre lucrările consacrate (la noi) în materie de baze de date, cea elaborată de un grup de profesori de la ASE Bucureşti condus de dl Ion Lungu, pentru DMV exemplul este corect[7], dar nu se prezintă nici o explicaţie sau definiţie a acesteia, adică de ce în acel exemplu R2{Curs, Capitol, CadruId} conţine o DMV. Iar câteva pagini mai încolo, în paragraful dedicat aducerii relaţiilor în 4NF[8], lucrurile se înceţoşează rău, cele două DMV prezentate la pagina 186 nefiind deloc convingătoare, cel puţin pentru mine.

Şi exemplul, altminteri clasic (curs – profesor titular – recomandare bibliografică), din [Oprea 99] este corect, inclusiv cele trei condiţii ale problemei, dintre care cea mai importantă este folosirea de către orice titular de curs a tuturor materialelor bibliografice folosite de ceilalţi titulari ai cursului respectiv[9]. Definiţia DMV, denumită de autor *dependenţă multiplă*, este însă destul de laxă: „O dependenţă multiplă există atunci când trei atribute se află într-o relaţie (notate A, B, C) şi pentru fiecare valoare a lui A există o mulţime bine definită de valori ale lui B şi alta bine definită de valori ale lui C. Totuşi, mulţimea valorilor lui B este independentă de mulţimea valorilor lui C"[10]. Or, sursa problemei este că noţiunea de independenţă a celor două mulţimi de valori nu este una obişnuită, după cum vom vedea peste nici jumătate de pagină.

Cea mai rezonabilă definiţie a DMV o furnizează C.J. Date. Fie relaţia R $\{A_1, A_2, \ldots, A_n\}$ şi X, Y şi Z trei subansambluri ale ansamblului $\{A_1, A_2, \ldots, A_n\}$. Există o dependenţă multivaloare între X şi Y în următoarea situaţie: *dacă (x,y,z) şi (x,y',z') sunt două tupluri ale relaţiei R, atunci (x,y',z) şi (x,y,z') aparţin, de asemenea, lui R.*

Într-o formulare uşor schimbată, Elmasri şi Navathe definesc DMV în maniera următoare: dacă în R există două tupluri t_1 şi t_2 pentru care $t_1[X] = t_2[X]$, atunci există în R alte două tupluri, t_3 şi t_4, care satisfac următoarele condiţii (notăm prin Z toate celelalte, decât X şi Y, atribute din R, adică (R – (X U Y))):

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$
- $t_3[Y] = t_1[Y]$ şi $t_4[Y] = t_2[Y]$
- $t_3[Z] = t_2[Z]$ şi $t_4[Z] = t_1[Z]$[11]

După cum afirmam mai sus, aspectul contra-intuitiv al definiţiei DMV ţine de faptul că X este independent de Y, ceea ce s-ar traduce prin „X oarecare, Y oarecare". O bună explicaţie în acest sens o găsim în [Dollinger 98][12]: o valoare dată a lui X se găseşte în R în combinaţie (formează tupluri) cu fiecare pereche de valori (y, z) din produsul cartezian al mulţimilor Yx (valorile lui y care apar în combinaţie cu un x dat) şi Zx. Aceasta înseamnă că mulţimile Yx şi Zx sunt independente între ele.

7. [Lungu ş.a. 95], pp. 167-168.
8. [Lungu ş.a. 95], pp. 185-186.
9. [Oprea 99], p. 352.
10. [Oprea 99], p. 353.
11. [Elmasri & Navathe 00], p. 514.
12. [Dollinger 98], p. 163.

Cele mai tentante exemple de DMV sunt cele teoretice, aşa cum este cel din figura 6.2.

XYZ

X	Y	Z
x1	y1	z2
x1	y2	z1
x1	y2	z2
x2	y2	z1
x2	y2	z2
x3	y1	z1
x1	y1	z1

Figura 6.2. Exemplu teoretic de relaţie în care există DMV

În această relaţie nu există nici una dintre dependenţele funcţionale:

X —/→ Y
X —/→ Z
Y —/→ X
Y —/→ Z
Z —/→ X
Z —/→ Y

La o examinare mai atentă se observă dependenţele multivaloare X ⟶→ Y (sau X ⟶→ Y | Z) şi X ⟶→ Z (sau X ⟶→ Z | Y); aceasta deoarece în condiţiile în care există tuplurile:

- (x1, y1, z1);
- (x1, y2, z2),

există şi tuplurile

- (x1, y1, z2) şi
- (x1, y2, z1).

Cel mai dureros aspect al dependenţelor multivaloare (şi al celor de joncţiune care vor urma) ţine de faptul că, în realitate, acestea trebuie identificate de către cei care fac analiza şi proiectarea bazei de date, pe baza legăturilor *semantice* dintre atributele bazei, şi nu pe baza unui caz simplist (în cazul nostru, o tabelă cu şapte linii).

Alţi copii, alte jucării

Asemănător relaţiei DOTARE_JUCĂRII_1 (figura 3.14), în figura 6.3 este prezentată tabela FCJ în care se evidenţiază jucăriile pe care le au copiii câtorva familii. Singura informaţie despre fiecare jucărie este numele său, astfel încât cheia primară a relaţiei este (Familie, Copil, Jucărie).

Iată restricţiile:

- o familie poate avea mai mult de un copil: Familie —/→ Copil;
- în orice familie pot exista mai multe jucării: Familie —/→ Jucărie;

FCJ

Familie	Copil	Jucărie
Popescu I Mihai	Ioana	Păpuşa "Barbie"
Popescu I Mihai	Răzvan	Tricicletă
Popescu I Mihai	Ioana	Tricicletă
Popescu I Mihai	Răzvan	Păpuşa "Barbie"
Vrânceanu V Vasile	Răzvan	Tricicletă
Vrânceanu V Vasile	Răzvan	Trenuleţ electric

Figura 6.3. Relaţia FCJ în care există DMV

- un copil poate avea mai multe jucării: `Copil` —/→ `Jucărie`;
- două sau mai multe familii pot avea copii cu acelaşi (pre)nume: `Copil` —/→ `Familie`;
- de asemenea: `Jucărie` —/→ `Copil`, `Jucărie` —/→ `Familie`.

Încercând a depista unele dependenţe funcţionale cu sursa compusă, vom avea aceleaşi rezultate:

(`Familie`, `Copil`) —/→ `Jucărie`
(`Familie`, `Jucărie`) —/→ `Copil`
(`Copil`, `Jucărie`) —/→ `Familie`

Dacă se aplică restricţia: o *aceeaşi jucărie trebuie cumpărată tuturor copiilor din familia respectivă (altfel iese scandal)*, avem de-a face cu o dependenţă multi-valoare:

`Familie` ⟶→ `Copil`

Deci, într-o familie oarecare, oricare ar fi copilul şi o jucărie, este sigur că oricare alt copil (al aceleiaşi familii) va avea această jucărie. Implicit, se verifică şi dependenţa multivaloare:

`Familie` ⟶→ `Jucărie`

De fapt, se poate scrie:

`Familie` ⟶→ `Copil` | `Jucărie` sau `Familie` ⟶→ `Jucărie` | `Copil`

Profesori, cursuri & capitole

Cazul următor este aproape identic celui prezentat de C.J. Date în ediţiile sale ale lucrării *An Introduction to Database Systems*[13]. Relaţia are trei atribute: CPC {`Curs`, `Profesor`, `Capitol`}[14]. La o facultate „oarecare" pot exista mai mulţi profesori titulari ai unei aceleiaşi discipline. Dacă presupunem că toţi profesorii cad de acord asupra unei programe unice pentru disciplina respectivă, diferenţiate fiind numai modul de „tratare" şi ponderea fiecărui capitol, putem imagina o situaţie ca în figura 6.4:

13. Spre exemplu, [Date 86], pp. 381-382.
14. La C.J. Date, relaţia era CTX {`Curs`, `Profesor`, `LucrareBibliografică`}.

CPC

Curs	Profesor	Capitol
Bazele informaticii economice	Fătu	Arhitectura microcalculatoarelor
Bazele informaticii economice	Airinei	Arhitectura microcalculatoarelor
Bazele informaticii economice	Grama	Arhitectura microcalculatoarelor
Bazele informaticii economice	Fătu	Generații de calculatoare
Bazele informaticii economice	Airinei	Generații de calculatoare
Bazele informaticii economice	Grama	Generații de calculatoare
Baze de date	Airinei	Modelul Entitate-Asociație
Baze de date	Fotache	SQL
Baze de date	Airinei	SQL
Baze de date	Fotache	Modelul Entitate-Asociație
Birotică	Grama	Poșta electronică

Figura 6.4. Relația CPC în care există DMV

Analizăm dependențele dintre atribute:
- un același curs poate fi „ținut" de mai mulți profesori: `Curs` —/→ `Profesor`;
- un curs este alcătuit din mai multe capitole: `Curs` —/→ `Capitol`;
- fiecare profesor poate preda mai multe cursuri: `Profesor` —/→ `Curs` și, implicit, `Profesor` —/→ `Capitol`;
- un același capitol poate apărea în cursuri diferite. De exemplu, un capitol dedicat Poștei electronice poate apărea și în cursul Bazele informaticii economice, și în cursul de Birotică. Capitolul Modelul Entitate-Asociație este în planul cursului de Baze de date, dar și al cursului Analiza sistemelor informatice (pentru modelul conceptual al datelor). Prin urmare, `Capitol` —/→ `Curs` iar `Capitol` —/→ `Profesor`.

Numai în condițiile unei programe unice, la nivelul facultății, pentru fiecare curs există dependențele multivaloare:

`Curs` ⟶→ `Profesor`
`Curs` ⟶→ `Capitol`

C.J. Date face o observație foarte bine venită: într-o asemenea relație, pentru a introduce trei profesori și patru capitole ar fi suficiente patru linii, și nu 12 (3*4). Aceasta naște însă câteva probleme: cum se realizează corespondența profesor--capitol, adică ce capitole a elaborat fiecare profesor? Numărul capitolelor fiind mai mare ca al autorilor, pentru acele linii, ce profesori apar? Cum este interpretată fiecare linie în acest caz? Cum se fac actualizările? Dar discuțiile acestea noi le-am purtat deja în capitolul 3.

Relația COMENZI_PRODUSE_MATERIALE

O întreprindere manufacturieră folosește pentru confecționarea unei unități de produs finit o serie de materii prime și materiale, în conformitate cu fișa tehnologică. Producția este pe comenzi. Se poate imagina o tabelă care să conțină informațiile cu privire la ce materiale sunt necesare onorării unei anumite comenzi, COMENZI_PRODUSE_MATERIALE - vezi figura 6.5.

COMENZI_PRODUSE_MATERIALE

Comandă	Produs	Material
1122	Produs 1	Material 2
1122	Produs 1	Material 4
1122	Produs 2	Material 1
1122	Produs 2	Material 3
1122	Produs 2	Material 4
1123	Produs 1	Material 2
1123	Produs 1	Material 4
1123	Produs 2	Material 1
1123	Produs 2	Material 3
1123	Produs 2	Material 4
1123	Produs 3	Material 2
1123	Produs 3	Material 4

Figura 6.5. Relaţia COMENZI_PRODUSE_MATERIALE

În această tabelă, este imperios ca, atunci când un produs apare într-o comandă, să se introducă câte o linie pentru fiecare material ce intră în fabricaţia produsului respectiv. Prin urmare, există dependenţele multivaloare:

`Produs ⟶→ Comandă | Material`

Din nou despre bibliotecă

Cel mai convingător exemplu de dependenţă multivaloare este cel din relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3 din figura 3.7, pe care o preluăm aşa cum am lăsat-o în figurile 5.21 (graful) şi 5.22 (conţinutul în 3NF-BCNF). În relaţia TITLURI_AUTORI_CUVINTECHEIE2, fiecare linie se referă la un autor şi un subiect (cuvânt-cheie) pentru o carte (`ISBN`). Practic, pe baza acestei tabele suntem în măsură să obţinem informaţii de genul:

- ce cărţi a scris un anume autor;
- despre ce subiecte a scris un anume autor;
- care sunt autorii unei cărţi;
- ce subiecte sunt tratate într-o carte;
- care sunt cărţile în care apare tratat un anumit subiect;
- care sunt autorii care au scris despre un subiect dat.

Pentru ca nici una dintre aceste informaţii să nu se piardă, există un preţ adecvat. Dacă, pentru o carte, apare o nouă sintagmă (cuvânt-cheie), în relaţie trebuie introdus nu un tuplu, ci *n*, unde *n* este egal cu numărul autorilor acelei cărţi. Pentru o carte cu 5 autori şi 10 cuvinte-cheie, în relaţia TITLURI_AUTORI_CUVINTECHEIE2 trebuie introduse nu mai puţin de 50 de linii.

Pe baza faptului că pentru o carte un autor trebuie să apară în combinaţie cu toate sintagmele tratate şi, viceversa, orice sintagmă apare în combinaţie cu toţi autorii cărţii, există DMV:

`ISBN ⟶→ Autor | CuvântCheie`
`ISBN ⟶→ CuvântCheie | Autor`

Prin analogie cu dependenţa funcţională, se poate defini *dependenţa multivaloare elementară* pentru cazul în care:

- $X \longrightarrow\!\!\!\!\rightarrow Y$;
- nu există nici un subansamblu X' şi Y' de atribute din X şi Y astfel încât $X' \longrightarrow\!\!\!\!\rightarrow Y'$.

Dependenţele multivaloare sunt o generalizare a celor funcţionale; orice DF este o DMV, reciproca nefiind valabilă (există DMV care nu sunt DF). Aceasta deoarece o DF este o DMV în care „setul" valorilor dependente corespunzătoare valorii unui determinant dat este întotdeauna o singură valoare[15].

Poate ar mai trebui evocate câteva exemple plauzibile pentru dependenţele multivaloare: cel al lui Robert Dollinger privind clienţii şi clientele unor agenţii matrimoniale – R {Agenţie, Nume_Femeie, Nume_Bărbat}, care însă, din păcate, nu este operaţional într-o serie de ţări mai „liberale" etc. În aceeaşi lucrare este un exemplu foarte bun legat de cursurile şi limbile străine în care pot fi predate, ţinând seama că orice profesor poate ţine fiecare curs pe care îl predă în toate limbile pe care le cunoaşte: R {Profesor, Disciplină, Limbă}.

6.2. A patra formă normală

Începeam acest capitol nu prea politicoşi cu privire la 4NF şi 5NF, exprimându-ne, ce-i drept, cât am putut de discret, rezervele legate de utilitatea celor două forme normale. Deşi multe lucrări, ca şi prezenta, le tratează la pachet, incidenţa lor practică este totuşi diferită. Aducem ca argument articolul publicat de Margaret Wu[16], în care autoarea afirmă că, în timp ce majoritatea universitarilor şi autorilor din zona normalizării tind să neglijeze 4NF şi 5NF, un studiu întreprins într-o serie de firme ce foloseau (la începutul anilor '90) aplicaţii cu baze de date a scos la iveală că mai mult de 20% dintre schemele bazelor de date analizate violau 4FN. Prin comparaţie, Wu nu a identificat nici un caz de violare a 5NF.

În lucrarea menţionată în paragraful anterior, Ronald Fagin formulează o teoremă conform căreia o relaţie R cu trei atribute A, B şi C, poate fi descompusă fără pierdere de informaţii în două proiecţii ale sale, R1{A, B} şi R2 {A, C}, dacă şi numai dacă există dependenţa multivaloare $A \longrightarrow\!\!\!\!\rightarrow B \mid C$. Aceasta este teorema care ne va permite aducerea unei relaţii în 4NF.

O relaţie este în 4NF dacă şi numai dacă:

1. este în BCNF;
2. toate dependenţele care se manifestă în cadrul său sunt dependenţe funcţionale, altfel spus, eventualele dependenţe multivaloare conţinute sunt, de fapt, dependenţe funcţionale.

Vom lua pe rând exemplele prezentate în paragraful anterior şi vom îndeplini formalitatea aducerii lor în 4NF. Dar, înainte de asta, trebuie să amintim faptul că

15. Pentru o demonstraţie *in extenso*, vezi şi [Garcia-Molina 02], pp. 121-122.
16. [Wu 92].

4NF a avut parte de un tratament similar DMV, iar confuziile nu sunt chiar izolate. Spre exemplu, în [Lungu ş.a. 95] există o definiţie pe care nu o putem trata cu prea mare îngăduinţă: o relaţie este în 4NF dacă în cadrul ei nu se manifestă mai mult de o dependenţă multivaloare[17]. Eu, unul, nu am priceput!

Începem cu cel mai îndrăgit exemplu, primul. Relaţia XYZ din figura 6.2 prezintă, aşa cum am discutat, dependenţa multivaloare $X \longrightarrow\!\!\!\!\rightarrow Y \mid Z$, în virtutea căreia o putem descompune în două: XY {X, Y} şi XZ {X, Z}, ca în figura 6.6.

XY

X	Y
x1	y1
x1	y2
x2	y2
x3	y1

XZ

X	Z
x1	z1
x1	z2
x2	z1
x2	z2
x3	z1

Figura 6.6. Aducerea relaţiei XYZ în 4NF

Pe acelaşi calapod, relaţia FCJ (figura 6.3) se descompune astfel încât cele două relaţii din figura 6.7 nu mai conţin dependenţe multivaloare (oricum, spre deosebire de alte lucrări, noi am mers pe ideea lui Date potrivit căreia, ca să existe o DMV, e nevoie de minimum trei atribute).

FC

Familie	Copil
Popescu I Mihai	Ioana
Popescu I Mihai	Răzvan
Vrânceanu V Vasile	Răzvan

FJ

Familie	Jucărie
Popescu I Mihai	Păpuşa "Barbie"
Popescu I Mihai	Tricicletă
Vrânceanu V Vasile	Tricicletă
Vrânceanu V Vasile	Trenuleţ electric

Figura 6.7. Relaţia FCJ în 4NF

Relaţia COMENZI_PRODUSE_MATERIALE

Pe baza dependenţelor multivaloare prezentate în paragraful precedent, relaţia COMENZI_PRODUSE_MATERIALE (figura 6.5) se descompune ca în figura 6.8.

COMENZI_PRODUSE

Comandă	Produs
1122	Produs 1
1122	Produs 2
1123	Produs 1
1123	Produs 2
1123	Produs 3

PRODUSE_MATERIALE

Produs	Material
Produs 1	Material 2
Produs 1	Material 4
Produs 2	Material 1
Produs 2	Material 3
Produs 2	Material 4
Produs 3	Material 2
Produs 3	Material 4

Figura 6.8. Aducerea relaţiei COMENZI_PRODUSE_MATERIALE în 4NF

17. [Lungu ş.a. 95], p. 170.

Relaţia BIBLIOTECĂ

Relaţia BIBLIOTECĂ_TUPLURI_NOI_SOLUŢIA_3, după cum am văzut în paragraful precedent, conţine o dependenţă multivaloare prin eliminarea căreia vom aduce relaţia în 4NF. Deşi rareori întâlnită în literatura de specialitate, reprezentarea grafică a DMV în graful dependenţelor nu ridică probleme deosebite, mai ales atunci când cele trei atribute (`X`, `Y` şi `Z` din definiţiile noastre) sunt simple. Iată, în figura 6.9, care ar fi o posibilă reprezentare şi modul de decupare a relaţiilor în 4NF din graf.

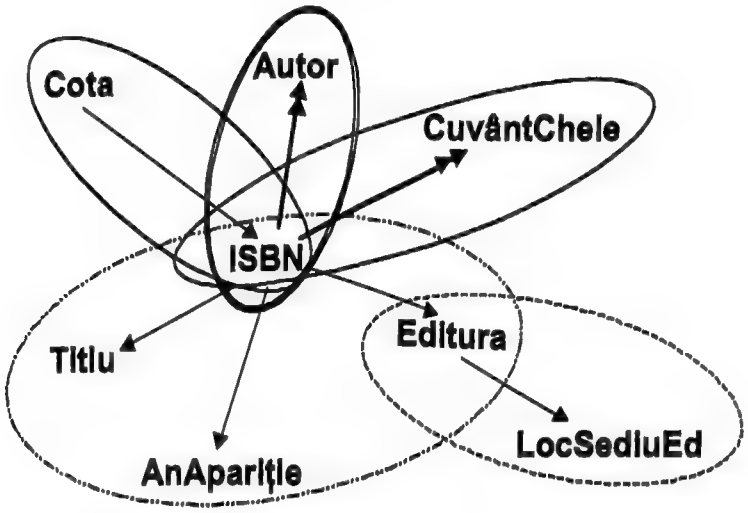


Figura 6.9. Graf ce conţine deopotrivă DF şi DMV

Conform obiceiului, se va constitui câte o relaţie pentru fiecare sursă de dependenţă funcţională (`Cota`, `ISBN` şi `Editura`). La aceste relaţii se va adăuga câte o relaţie pentru fiecare destinaţie de dependenţă multivaloare (`Autor` şi `CuvântCheie`). Schema finală a bazei de date, precum şi conţinutul relaţiilor sunt ilustrate în figura 6.10.

Iată că, după atâta amar de capitole şi discuţii, am reuşit să rezolvăm într-o manieră acceptabilă problema schemei bazei de date destinate gestionării informaţiilor revelante despre cărţile aflate în biblioteca unei facultăţi/universităţi. Este, cred, inutil să mai intrăm în delalii vizavi de reducerea volumului bazei, ca şi cvasi-dispariţia oricărei forme de anomalie la inserarea, modificarea sau ştergerea unei linii în oricare dintre cele cinci tabele.

Baza de date FILMOGRAFIE

Discuţiile despre această bază de date au rămas suspendate încă din capitolul 3. În paragraful 3.4 semnalam necazurile pricinuite de valorile nule ale cheii primare în condiţiile în care o parte dintre premii erau acordate la nivel de film (*Oscar pentru regie*, *Oscar pentru cel mai bun film* etc.), iar altă parte la nivel de actor pentru prestaţia dintr-un film (*Oscar pentru cel mai bun actor/cea mai bună actriţă în rol principal/secundar*). La acel moment, singura soluţie era constituirea a două tabele dedicate premiilor.

Pentru ceea ce urmează considerăm atributele din tabelul 3.1 (paragraful 3.2), între care încercăm să determinăm dependenţele funcţionale şi multivaloare. Astfel, atributul `IdFilm` identifică fără ambiguitate orice film, aşa că avem următoarea serie de DF:

EDITURI

Editura	LocSediuEd
Polirom	Iaşi

TITLURI

ISBN	Titlu	Editura	AnApariţie
973-683-889-7	Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale	Polirom	2002
973-683-709-2	SQL. Dialecte DB2, Oracle şi Visual FoxPro	Polirom	2001

EXEMPLARE

Cotă	ISBN
III-13421	973-683-889-7
III-13422	973-683-889-7
III-13423	973-683-889-7
III-10678	973-683-709-2
III-10679	973-683-709-2

TITLURI_AUTORI

ISBN	Autor
973-683-889-7	Marin Fotache
973-683-889-7	Ioan Brava
973-683-889-7	Cătălin Strâmbei
973-683-889-7	Liviu Creţu
973-683-709-2	Marin Fotache

TITLURI_CUVINTECHEIE

ISBN	CuvântCheie
973-683-889-7	baze de date
973-683-889-7	SQL
973-683-889-7	proceduri stocate
973-683-889-7	FoxPro
973-683-889-7	formulare
973-683-889-7	orientare pe obiect
973-683-889-7	client-server
973-683-889-7	Web
973-683-709-2	baze de date
973-683-709-2	algebră relaţională
973-683-709-2	SQL

Figura 6.10. 4NF pentru relaţia BIBLIOTECĂ

(1) `IdFilm` ⟶ `TitluOriginal`
(2) `IdFilm` ⟶ `TitluRO`
(3) `IdFilm` ⟶ `AnLans`

Deoarece un film poate fi încadrat în mai multe genuri (vezi figura 2.5) şi poate avea mai mulţi producători, regizori, actori şi premii:

`IdFilm` —/→ `Gen`
`IdFilm` —/→ `Producător`
`IdFilm` —/→ `Regizor`
`IdFilm` —/→ `Actor`
`IdFilm` —/→ `DenPremiu`

Un premiu se decernează într-o localitate (de exemplu: *Oscarul* la Hollywood, *Leul de argint* la Veneţia, *Ursul de aur* la Berlin etc.):

(4) `DenPremiu` ⟶ `LocDecernare`

Premiile au, de obicei, mai multe categorii (de exemplu : Oscarul are categoriile : *Cel mai bun film*, *Cel mai bun film străin*, *Efecte speciale*, *Regie* etc.) şi se acordă de mulţi ani :

`DenPremiu` —/→ `Categorie`
`DenPremiu` —/→ `AnPremiu`

Ne punem problema reflectării prin dependenţe a distribuţiei într-un film. După cum am văzut mai sus (de fapt, ştiam de mult), într-un film sunt distribuiţi mai mulţi actori ; însă şi un actor poate juca în mai multe filme - `Actor` —/→ `IdFilm`. Am avea de ales între dependenţele :

(`IdFilm`, `Actor`) ⟶ `Rol`
(poate era mai nimerit ca atributul să se numească `Personaj`) sau
(`IdFilm`, `Rol`) ⟶ `Actor`.

Dumneavoastră pe care aţi alege-o ? Ei bine, prima n-ar fi chiar indicată, deoarece există filme în care un actor joacă un dublu rol (vă mai amintiţi *Zmeura de aur* pentru dublul rol al lui Leonardo di Caprio din filmul *Masca de fier*, în care îl interpretează pe regele Ludovic al XIV-lea şi pe fratele regelui, unul „original", iar celălalt „mascatul" ?). În aceste situaţii valorile grupului (`IdFilm`, `Actor`) sunt identice, în timp ce valorile atributului `Rol` sunt diferite, ceea ce infirmă dependenţa. Privitor la cea de-a doua dependenţă, pericolul invalidării ei ar apărea în situaţia în care într-un film acelaşi rol este jucat de mai mulţi actori. Din fericire, rolurile nu sunt tocmai aşa. Chiar şi în filmele biografice, sau marile saga de familie, diferitele etape din viaţa unui personaj (copil, adolescent etc.) sunt marcate de roluri distincte. Aşa că numărăm a cincea dependenţă a relaţiei.

(5) (`IdFilm`, `Rol`) ⟶ `Actor`

Cel puţin la fel de interesant stau lucrurile şi cu premiile. Faptul că *Globul de aur* (`DenPremiu`) pentru *Cea mai bună imagine* (`Categorie`) a fost decernat în 1988 (`AnPremiu`) filmului *Mai bine nu se poate* (titlul original - *As Good As It Gets*), film ce are codul (`IdFilm`) *11899* în baza noastră de date, ne-ar determina să scriem dependenţa în forma : (`DenPremiu`, `Categorie`, `AnPremiu`) ⟶ `IdFilm`.

Ce ne facem însă când un premiu la aceeaşi categorie este acordat simultan pentru două filme ? Nu e cazul Oscarurilor, dar, spre exemplu, câte o *menţiune specială din partea juriului* (`Categorie`) la ediţia din 2007 (`AnPremiu`) a festivalului *Ursul de aur* (`Denpremiu`) ar putea fi acordată filmelor cu identificatorii *7898843*, *8967883* şi *9312299*. Dependenţa este aruncată în aer. Soluţia este însă pe-aproape : un film nu poate lua un acelaşi premiu, la aceeaşi categorie, în ani diferiţi, aşa că schimbăm poziţia a două atribute :

(6) (`DenPremiu`, `Categorie`, `IdFilm`) ⟶ `AnPremiu`.

Dependenţa de mai sus nu rezolvă problema premiilor acordate pentru prestaţiile actoriceşti, în care laureat este un actor pentru prestaţia dintr-un film. Am putea alege între varianta :

(`DenPremiu`, `Categorie`, `AnPremiu`, `IdFilm`) ⟶ `Actor`
şi :
(`DenPremiu`, `Categorie`, `AnPremiu`, `Actor`) ⟶ `IdFilm`.

Prima variantă este minată dacă într-un an un premiu (la aceeaşi categorie) este acordat simultan pentru doi actori care joacă în acelaşi film. Situaţia nu pare a fi prea frecventă, dar nici imposibilă, dacă ne gândim la festivalurile mai puţin poleite decât *Oscar*, *Cannes* etc. A doua variantă are şi ea un competitor, croit după analogia cu dependenţa (6) : (`DenPremiu, Categorie, IdFilm, Actor`) ⟶ `AnPremiu`. Iniţial eram tentat de această ultimă formă, deoarece era similară celei care reflecta celelalte tipuri de premii. Există însă o categorie de premii ce creează o serie de dureri de cap : premiile pentru întreaga carieră. Acestea nu sunt legate de un film anume, ci răsplătesc o carieră, mai lungă sau mai scurtă, de cineast. Nefiind legată de un film, sursa dependenţei, pe baza căreia vom constitui o relaţie, ar conţine în aceste situaţii o valoare NULL (pentru `IdFilm`), ceea ce e inadmisibil pentru o componentă a cheii primare. Cu toate acestea, vom păstra această dependenţă, întrucât premiul pentru întreaga carieră nu se acordă numai actorilor, ci şi regizorilor (teoretic, şi altor profesiuni legate de industria/arta cinematografică), iar dacă optam pentru a doua variantă, tot nu scăpam de valorile nule, e drept, pentru un atribut non-cheie. Ca să tranşăm discuţia, întrucât baza de date se referă la informaţii despre filme, nu ne propunem să preluăm premiile individuale, aşa că ne declarăm mulţumiţi de dependenţa :

(7) (`DenPremiu, Categorie, IdFilm, Actor`) ⟶ `AnPremiu`.

Au rămas neluate în seamă trei atribute, `Producător`, `Regizor` şi `Gen`. Intuiţia ne spune că este imposibil de stabilit vreo dependenţă între ele, chiar dacă se referă la acelaşi film. Pe de altă parte, cele trei atribute par cât se poate de independente unele de altele. Verificăm, mai întâi, cerinţa DMV pentru primele două. Dacă filmul X cu ID-ul *234556* are, printre regizori, pe *Reg1* şi *Reg2* şi cel puţin doi producători, *Prod1* şi *Prod2*, atunci trebuie să verificăm că, dacă atributele ar apărea într-o relaţie de tip R {`IdFilm, Producător, Regizor`}, oricare dintre tuplurile următoare este posibil :

RELx

IdFilm	Producător	Regizor
...	...	...
234556	Prod1	Reg1
234556	Prod2	Reg2
234556	Prod1	Reg2
234556	Prod2	Reg1
...	...	...

Figura 6.11. Raportul dintre atributele `Producător` şi `Regizor`

Prezenţa ultimelor două tupluri în relaţia din figura 6.11 pare a fi agreată, nu însă şi necesară. Schema de mai sus însă trebuie să furnizeze răspunsul corect la întrebarea *Care sunt filmele în care producătorul Prod1 a lucrat cu regizorul Reg1* ? Dacă ultimele două tupluri nu ar fi prezente în relaţie, atunci rezultatul furnizat de interogarea :

```
SELECT IdFilm
FROM RELx
WHERE Producător = 'Prod1' AND Regizor='Reg1'
```

este incorect, singura soluţie acceptabilă fiind cea „ocolitoare":

```
SELECT IdFilm FROM RELx WHERE Producător = 'Prod1'
INTERSECT
SELECT IdFilm FROM RELx WHERE Regizor='Reg1'
```

sau, fireşte:

```
SELECT IdFilm FROM RELx WHERE Producător = 'Prod1'
AND IdFilm IN
(SELECT IdFilm FROM RELx WHERE Regizor='Reg1')
```

Faptul că, pentru a-şi păstra viabilitatea informaţională, relaţia trebuie să conţină toate cele patru tupluri constituie un motiv suficient pentru a „trâmbiţa" dependenţa multivalorică:

IdFilm ⟶→ Producător | Regizor

Pe acelaşi calapod, pornind de la răspunsul la întrebarea *Ce genuri de filme a produs/regizat un anumit cineast ?*, putem demonstra că:

IdFilm ⟶→ Producător | Gen sau IdFilm ⟶→ Regizor | Gen

Forma finală a grafului dependenţelor funcţionale şi multivalorice ar putea fi cea din figura 6.12.

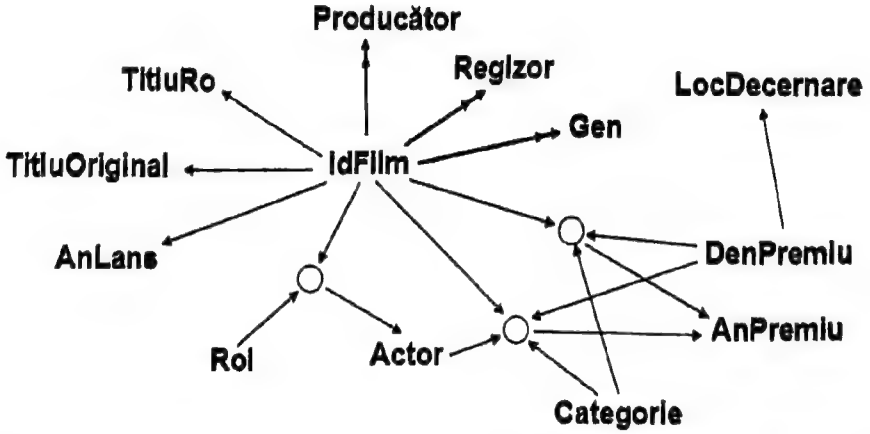


Figura 6.12. Graful dependenţelor pentru BD FILMOGRAFIE

Urmează partea cea mai frumoasă – decuparea relaţiilor din graf. Sunt două surse simple de DF, DenPremiu şi IdFilm, şi trei surse compuse, (IdFilm, Rol), (IdFilm, DenPremiu, Categorie) şi (IdFilm, Actor, DenPremiu, Categorie), aşa că cinci relaţii vor fi obţinute din dependenţele funcţionale. La acestea se mai adaugă trei, corespunzătoare dependenţelor multivaloare. Iată, deci, schema finală:

PREMII_DENUMIRI {DenPremiu, LocDecernare}
FILME {IdFilm, TitluOriginal, TitluRO, AnLans}

DISTRIBUŢIE {IdFilm, Rol, Actor}
PREMII_FILME {IdFilm, DenPremiu, Categorie, AnPremiu}
PREMII_INTERPRETARE {IdFilm, Actor , DenPremiu, Categorie, AnPremiu}
PRODUCĂTORI {IdFilm, Producător}
REGIZORI {IdFilm, Regizor}
FILME_GENURI {IdFilm, Gen}

Revenind preţ de o frază la discuţia legată de producători/regizori/genuri, acum, răspunsul la întrebarea *Care sunt filmele în care producătorul Prod1 a lucrat cu regizorul Reg1?* poate fi obţinut doar parcurgând ambele relaţii, PRODUCĂTORI şi REGIZORI, iar corectitudinea rezultatului este garantată:

```
SELECT IdFilm FROM PRODUCĂTORI WHERE Producător = 'Prod1'
INTERSECT
SELECT IdFilm FROM REGIZORI WHERE Regizor='Reg1'.
```

Relaţia DOTARE_JUCĂRII

Până aici, aducerea în 4NF a unei baze de date prin identificarea şi eliminarea dependenţelor multivaloare nu a generat chestiuni insurmontabile, iar operaţiunile se simplifică şi mai mult dacă se recurge la graful dependenţelor funcţionale şi multivaloare. Lucrurile se pot complica fără prea mare efort dacă în relaţia DOTARE_JUCĂRII_1, reprezentată în figura 3.14, ce are cheie primară combinaţia (CodFamilie, Copil, Jucărie), presupunem că, într-o familie, orice jucărie este cumpărată în *n* exemplare, unde *n* reprezintă numărul copiilor din acea familie (prin urmare, se verifică dependenţa multivaloare dintre CodFamilie şi Copil/Jucărie). Graful celor două tipuri de dependenţe, cel din figura 6.13, este diferit în funcţie de modalitatea de cumpărare a celor *n* exemplare ale fiecărei jucării.

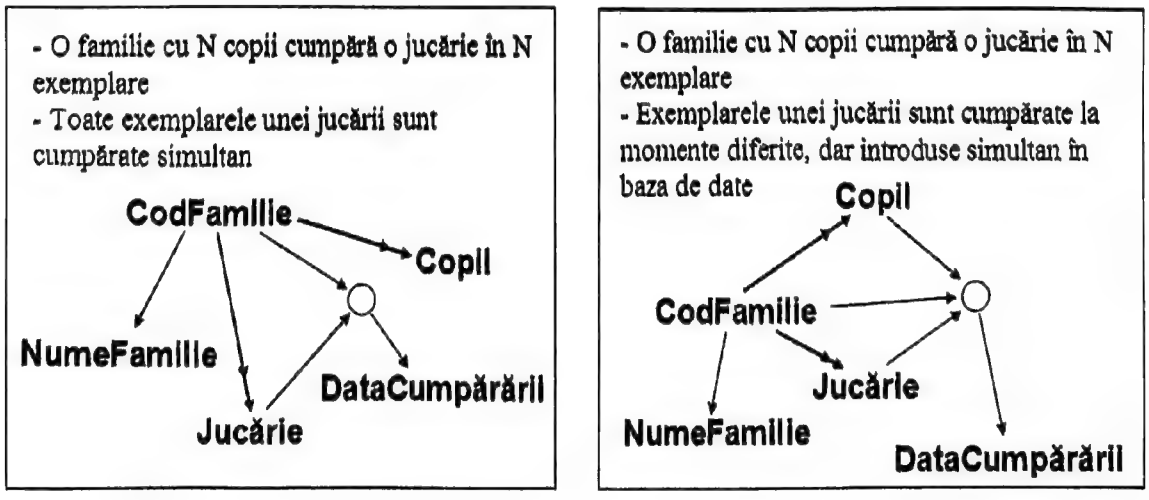


Figura 6.13. Graful dependenţelor pentru DOTARE_JUCĂRII în cele două situaţii

Despre cele două grafuri nu se poate spune că sunt din cale-afară de simple. Să începem cu cel din partea stângă. Există DMV CodFamilie $\longrightarrow\!\!\!\!\rightarrow$ Jucărie | Copil,

însă sursa şi una dintre destinaţiile DMV formează sursa unei dependenţe funcţionale : (CodFamilie, Jucărie) ⟶ DataCumpărării. De aceea, este rezonabil ca, la descompunerea ce decurge din dependenţa multivaloare, să includem DataCumpărării alături de Jucărie (relaţia JUCĂRII). Schema 4NF a bazei ar fi în acest caz :

FAMILII {CodFamilie, NumeFamilie}
COPII {CodFamilie, Copil}
JUCĂRII {CodFamilie, Jucărie, DataCumpărării}

A doua situaţie este cu mult mai interesantă. Din raţiuni financiare, o familie cu patru copii poate cumpăra trei trenuleţe într-o zi, iar al patrulea trenuleţ la chenzina următoare. Abia după achiziţia celui de-al patrulea se preiau în bază toate trenuleţele. Chiar dacă s-au jucat, clandestin sau nu, cu trenuleţele înainte de completarea „seriei", copiii ştiu cu exactitate cui aparţine fiecare dintre cele patru trenuleţe. În acest caz, singura dependenţă funcţională cu sursa compusă este cea care decurge din calitatea de cheie primară a combinaţiei (CodFamilie, Copil, Jucărie). Ei bine, deşi, conform obiceiului, atunci când într-o relaţie există o dependenţă multivaloare relaţia se poate descompune (aşa cum am făcut în acest paragraf), dacă am opta pentru schema de mai sus - FAMILII-COPII-JUCĂRII - am pierde o informaţie importantă : data la care a fost cumpărată jucăria unui anumit copil. Concluzia este deconcertantă : nu întotdeauna când într-o relaţie avem o dependenţă multivaloare ruperea sa reprezintă soluţia cea mai indicată, fiind necesară verificarea pierderii sau nu de informaţii la descompunere ! Schema prin care nu pierdem informaţii este :

FAMILII {CodFamilie, NumeFamilie}
COPII {CodFamilie, Copil}
JUCĂRII_FAMILII {CodFamilie, Jucărie }
JUCĂRII_COPII {CodFamilie, Jucărie, Copil, DataCumpărării}

Redundanţa pare supărătoare, însă nu insuportabilă. În JUCĂRII_FAMILII, fiecare tip de jucărie apare o singură dată la o familie, în timp ce în JUCĂRII_COPII îi sunt alocate *n* linii. Totuşi, în condiţiile date, JUCĂRII_FAMILII este o relaţie redundantă. Şi relaţia COPII este una redundantă, singurul său avantaj fiind că elimină o importantă anomalie la inserare, şi anume, se permite inserarea în baza de date a unuia sau mai multor copii dintr-o aceeaşi familie, chiar dacă în acea familie nu a fost cumpărată nici o jucărie.

Vom vedea în capitolul viitor că o mare parte din confuzia problemei ţine de faptul că nici jucăriile, nici copiii nu sunt identificabili prin cele două atribute, Jucărie şi Copil, iar folosirea cheilor surogat mai împrăştie din ceaţă.

6.3. Dependenţe de joncţiune

Dacă dependenţele multivaloare constituie o raritate în proiectarea bazelor de date, atunci ce să mai vorbim despre cele de joncţiune ? Condiţiile ce trebuie întrunite pentru a „depista" o dependenţă de joncţiune şi, implicit, a pune problema aducerii bazei

de date în a cincea formă normalizată sunt atât de restrictive, încât, într-o mare parte din lucrările dedicate bazelor de date sau proiectării bazelor de date, 5NF nici nu se mai discută[18], exemplele sunt strict teoretice[19], insuficient explicate[20] sau sumare[21].

Dependența de joncțiune definită în continuare este legată de numele lui Jorma Rissanen[22] și folosește definiția generalizată a joncțiunii unei colecții de relații formulată de Aho, Beeri și Ullman[23]. Fie R $(A_1, A_2, \ldots, A_n)$ o relație și $X_1, X_2, \ldots, X_m$, m subansambluri de atribute din R. Se spune că există o dependență de joncțiune de ordinul m între $X_1, X_2, \ldots, X_m$, notată:

$$* X_1 / X_2 / \ \ldots \ / X_m \text{ sau } * (X_1, X_2, \ldots, X_m)$$

dacă și numai dacă R reprezintă joncțiunea proiecțiilor sale pe $X_1, X_2, \ldots, X_m$, altfel spus:

$$R = R(X_1) * R(X_2) * \ldots * R(X_m)$$

unde * reprezintă o (altă) notație pentru operatorul de joncțiune internă, iar $R(X_i)$ este proiecția lui R pe X_i.

Când $m=3$, dependența este una *mutuală*[24], în timp ce pentru $m=2$ este vorba de o dependență multivaloare.

Fie R (X_1, X_2, X_3) o relație (X_i – subansamblu de atribute). X_1 și X_2 sunt disjuncte, iar X_3 este complementul față de R al $(X_1 \cup X_2)$. Se spune că există o *dependență mutuală* între X și Y dacă și numai dacă, atunci când (X_1, X_2, X_3) și (X'_1, X'_2, X'_3) sunt două tupluri ale relației $R(X_1, X_2, X_3)$:

- $(X_2, X'_3) \in R(X_2, X_3)$, atunci $(X_1, X_2, X'_3) \in R(X_1, X_2, X_3)$;
- $(X'_2, X_3) \in R(X_2, X_3)$, atunci $(X_1, X'_2, X_3) \in R(X_1, X_2, X_3)$.

Se poate scrie:

$$R(X_1, X_2, X_3) = R(X_1, X_2) * R(X_1, X_3) * R(X_2, X_3).$$

Primul exemplu

Pornim de la relația XYZ, alcătuită din tuplurile din figura 6.14.

XYZ

X	Y	Z
x1	y1	z2
x1	y2	z1
x2	y1	z1
x1	y1	z1

Figura 6.14. Relația XYZ în care există dependențe de joncțiune

18. Spre exemplu, [Garcia-Molina 02], [Florescu ș.a. 99], [Connoly & Begg 00].
19. [Lungu ș.a. 95], [Băscă 97].
20. [Celko 99].
21. [Riordan 99], [Fleming & von Halle 89], [Teorey 99].
22. [Rissanen 77].
23. [Aho ș.a. 79].
24. Sintagma *dependență mutuală* a fost introdusă în [Nicolas 78].

Se „sparge" relaţia de mai sus în trei relaţii ce reprezintă proiecţiile, pe câte două atribute, ale acesteia – figura 6.15.

XY

X	Y
x1	y1
x1	y2
x2	y1

YZ

Y	Z
y1	z2
y2	z1
y1	z1

XZ

X	Z
x1	z2
x1	z1
x2	z1

Figura 6.15. Proiecţiile, pe câte două atribute, ale relaţiei XYZ

Operaţia XY_YZ ⟵ JONCŢIUNE (XY, YZ; XY.*Y*=YZ.*Y*) conduce la rezultatul din figura 6.16, iar prin joncţionarea XY_YZ cu XZ se obţine o relaţie cu aceleaşi linii ca şi XYZ.

XY_YZ

X	Y	Z
x1	y1	z2
x1	y1	z1
x1	y2	z1
x2	y1	z2
x2	y1	z1

Figura 6.16. Tabela XY_YZ

Prin urmare, se poate scrie:

$$XYZ = XY * YZ * XZ$$

După Date[25], afirmaţia *XYZ este egală cu joncţiunea celor trei proiecţii ale sale, XY, XZ şi YZ* este sinonimă cu a demonstra că, dacă perechea (x1, y1) apare în XY, (y1, z1) apare în YZ şi (x1, z1) apare în XZ, *atunci (x1, y1, z1) apare în XYZ*. Tot el formulează condiţia pentru dependenţa de joncţiune şi astfel: dacă în XYZ există trei tupluri, (x1, y1, z2), (x2, y1, z1) şi (x1, y2, z1), atunci obligatoriu apare şi un al patrulea, (x1, y1, z1).

Agenţi, produse, companii

Pornind de la un exemplu preluat din lucrarea lui Nicolas[26], Ronald Fagin ilustrează într-unul dintre articolele sale[27] dependenţa de joncţiune cu o relaţie de genul R {`Agent, Produs, Companie`}. Un tuplu al acestei relaţii, spre exemplu, (*Vasilescu, Detergent Spuma Mării, Hurstman*), semnifică faptul că agentul *Vasilescu* reprezintă detergentul *Spuma Mării* produs de compania *Hurstman*. În R există o dependenţă de joncţiune dacă şi numai dacă, presupunând că:

- agentul *a* reprezintă produsul *p* fabricat de compania *c'*;
- agentul *a* reprezintă produsul *p'* fabricat de compania *c* şi că

25. [Date 86], p. 387.
26. [Nicolas 78].
27. [Fagin 81].

– agentul *a'* reprezintă produsul *p* fabricat de compania *c*,
atunci *agentul a reprezintă produsul p fabricat de compania c*!

Relaţia PCS

Pornim de la relaţia PCS prezentată în figura 6.17, în care este reflectată, parţial, situaţia cursurilor de informatică la specializări ale Facultăţii de Economie şi Administrarea Afacerilor (situaţie de înainte de 1998, deoarece acum, primii doi ani sunt comuni la profilul Economic).

PCS

Profesor	Curs	Specializare
Fătu	Bazele informaticii economice	Finanţe
Fătu	Bazele informaticii economice	Contabilitate
Filip	Bazele informaticii economice	Finanţe
Filip	Baze de date	Finanţe
Filip	Bazele informaticii economice	Management
Filip	Baze de date	Management
Airinei	Baze de date	Marketing
Airinei	Bazele informaticii economice	Marketing
Airinei	Bazele informaticii economice	Administraţie Publică
Airinei	Baze de date	Management
Airinei	Bazele informaticii economice	Management

Figura 6.17. Relaţia PCS aflată în 4NF, nu însă şi în 5NF

Comentarii:

- un profesor predă mai multe cursuri: `Profesor` —/→ `Curs`;
- un profesor predă (unul sau mai multe cursuri) la mai multe specializări: `Profesor` —/→ `Specializare`;
- un acelaşi curs poate fi ţinut de mai mulţi profesori: `Curs` —/→ `Profesor`;
- un acelaşi curs poate fi în planul de învăţământ al mai multor specializări: `Curs` —/→ `Specializare`;
- la o specializare, în planul de învăţământ sunt incluse mai multe cursuri: `Specializare` —/→ `Curs`;
- la o specializare ţin cursuri mai mulţi profesori: `Specializare` —/→ `Profesor`;
- la o specializare, un acelaşi curs poate fi predat de mai mulţi profesori. Explicaţie: la unele specializări, în cadrul aceluiaşi an de studii există doi semiani de predare; prin urmare, (`Specializare, Curs`) —/→ `Profesor`;
- un profesor poate ţine un acelaşi curs la mai multe specializări: (`Profesor, Curs`) —/→ `Specializări`;
- la o specializare, un acelaşi profesor poate ţine mai multe cursuri: (`Specializare, Profesor`) —/→ `Curs`.

Până în acest moment, am demonstrat inexistenţa oricărei dependenţe funcţionale, simple sau compuse. În continuare verificăm existenţa vreunei dependenţe multivaloare. Există tuplurile:

(*Airinei, Bazele informaticii economice, Administraţie Publică*)

şi
(*Airinei*, *Baze de date*, *Management*).

Dacă ar opera dependenţa multivaloare `Profesor` —/→ `Curs | Specializare`, ar trebui ca în relaţia PCS să existe şi tuplurile:

(*Airinei*, *Bazele informaticii economice*, *Management*)
şi
(*Airinei*, *Baze de date*, *Administraţie Publică*).

Or, ultimul tuplu nu există, deoarece cursul *Baze de date* nu este în planul de învăţământ al specializării *Administraţie Publică*. Rezultă că în PCS nu există nici o dependenţă multivaloare. Nu ne mai rămâne decât să dovedim, cu ultimele puteri, că PCS, adică R(`Profesor, Curs, Specializare`) = R(`Profesor, Curs`) * R(`Profesor, Specializare`) * R(`Curs, Specializare`).

Mai întâi, din PCS obţinem tabelele PC, PS şi CS, prin proiecţiile pe câte două atribute ale acestei relaţii – vezi figura 6.18.

PC

Profesor	Curs
Fătu	Bazele informaticii economice
Filip	Bazele informaticii economice
Filip	Baze de date
Airinei	Baze de date
Airinei	Bazele informaticii economice

PS

Profesor	Specializare
Fătu	Finanţe
Fătu	Contabilitate
Filip	Finanţe
Filip	Management
Airinei	Marketing
Airinei	Administraţie Publică
Airinei	Management

CS

Curs	Specializare
Bazele informaticii economice	Finanţe
Bazele informaticii economice	Contabilitate
Baze de date	Finanţe
Bazele informaticii economice	Management
Baze de date	Management
Baze de date	Marketing
Bazele informaticii economice	Marketing
Bazele informaticii economice	Administraţie Publică

Figura 6.18. Proiecţiile pe câte două atribute ale relaţiei PCS

Comentarii:

- Prof. Fătu ţine cursul de *Bazele informaticii economice*, prof. Fătu are ore la specializarea *Finanţe*, iar disciplina *Bazele informaticii economice* apare în planul de învăţământ la specializarea *Finanţe* şi

prof. Fătu ţine cursul de Bazele Informaticii Economice
la specializarea Finanţe.

- Prof. Fătu ţine cursul de *Bazele informaticii economice*, prof. Fătu are ore la specializarea *Contabilitate*, iar *Bazele informaticii economice* este în planul de învăţământ al specializării *Contabilitate* şi

prof. Fătu ţine cursul de Bazele Informaticii Economice la specializarea Contabilitate.

- Prof. Filip ţine cursul de *Bazele informaticii economice*, prof. Filip are ore la specializarea *Finanţe*, iar *Bazele informaticii economice* este în planul de învăţământ la specializarea *Finanţe* şi

prof. Filip ţine cursul de Bazele Informaticii Economice la specializarea Finanţe.

Generalizând:

- dacă profesorul P ţine cursul C,
- profesorul P ţine ore la specializarea S şi
- cursul C este în planul de învăţământ al specializării S,

atunci

profesorul P ţine cursul C la specializarea S.

Numai în aceste condiţii există dependenţa de joncţiune (mutuală, în acest caz)

* {(*Profesor*, *Curs*), (*Profesor*, *Specializare*), (*Curs*, *Specializare*)}.

Vânzări şi credite

De câţiva ani a început şi la noi vânzarea bunurilor de folosinţă îndelungată (maşini de spălat, frigidere, televizoare etc.) cu plata în rate. Magazinele încheie în prealabil acorduri cu una sau mai multe bănci pentru acordarea creditelor pentru bunurile vândute. La vânzare, clienţilor li se prezintă băncile la care pot apela pentru credite. Clienţii decid asupra băncii celei mai convenabile şi, astfel, vânzarea-i gata (cu rambursarea e altă discuţie, mult mai interesantă). Relaţia cumpărător-magazin-bancă este reflectată în tabela CREDITE_RATE din figura 6.19.

CREDITE_RATE

Cumpărător	Magazin	Bancă
Alexa V. Gheorghe	Onitrade Ltd.	BRD
Alexa V. Gheorghe	Domo	BCR
Georgescu P. Mircea	Onitrade Ltd.	Banca Ion Ţiriac
Brava H. Ion	Moldova	Banca Ion Ţiriac
Brava H. Ion	Onitrade Ltd.	Banca Ion Ţiriac
Brava H. Ion	Onitrade Ltd.	BRD
Fînaru W. Luminiţa	Onitrade Ltd.	Banca Ion Ţiriac
Fînaru W. Luminiţa	Domo	BCR
Fînaru W. Luminiţa	Moldova	Banca Albina

Figura 6.19. Relaţia protagoniştilor pentru vânzările în rate

Să presupunem că sunt valabile următoarele restricţii:

- dacă un magazin (firmă) *M* a stabilit un acord cu o bancă *B* privind vânzări în rate;
- dacă un client *C* cumpără bunuri de la magazinul *M*;
- dacă clientul *C* a ales să ia credit de la banca *B*;

atunci *pentru bunurile achiziţionate de la magazinul M clientul C va apela la serviciile băncii B.*

În aceste condiţii, există o dependenţă de joncţiune notată:

* { CumpărătorMagazin, MagazinBancă, CumpărătorBancă }

sau

* { { Cumpărător, Magazin }, { Magazin, Bancă }, { Cumpărător, Bancă } }

Ca şi în cazul dependenţelor multivaloare şi 4NF, lucrarea românească (din câte cunosc eu) în care se explică cel mai bine dependenţele de joncţiune şi 5NF este cea a lui Robert Dollinger. Exemplul său este relaţia PDL {Profesor, Disciplină, Limbă}, iar condiţia în care se manifestă dependenţele de joncţiune (autorul le numeşte *dependenţe de cuplare*[28]) este: *Dacă o anumită disciplină este predată într-o anumită limbă, atunci acest lucru este făcut de fiecare profesor care predă disciplina respectivă şi cunoaşte limba în cauză*[29].

Şi exemplul lui Riordan este sugestiv. Dacă într-o relaţie R{Furnizor, Produs, Client} se verifică restricţia: *atunci când un furnizor F vinde un produs P, un client C cumpără acel produs P, iar furnizorul F are o relaţie (comercială) cu clientul C (în sensul că F a vândut măcar un produs clientului C), obligatoriu furnizorul F va vinde produsul P clientului C.* Ulterior, aveam să descopăr că exemplul este preluat din [Fleming & von Halle 89].

6.4. A cincea formă normală

Cea de-a cincea formă normală – denumită şi *forma normală proiecţie-joncţiune* (*Project-Join Normal Form*) – are la bază dependenţa de joncţiune şi a fost definită tot de Ronald Fagin[30]. *O relaţie R este în 5NF dacă şi numai dacă toate dependenţele de joncţiune sunt consecinţe ale cheilor candidate ale lui R*[31].

De remarcat că 5NF, 4NF şi BCNF corespund definiţiei unice: *orice determinant al unei dependenţe este o cheie*; ceea ce diferă ţine de faptul că pentru BCNF este vorba de dependenţă *funcţională*, pentru 4NF – dependenţă *multivaloare*, iar pentru 5NF – *dependenţă de joncţiune*. 5NF este o generalizare a 4NF, care este o generalizare a BCNF.

28. Probabil că dacă era moldovean (ca mine) le-ar fi zis *dependenţe de alăturare*.
29. [Dollinger 98], p. 174.
30. [Fagin 79].
31. [Date 86], p. 389.

Până la 5NF, o relaţie era descompusă prin proiecţie în două relaţii ce puteau, prin joncţiune naturală, să „refacă" în orice moment relaţia iniţială. Cu dependenţa de joncţiune, această descompunere fără pierderi poate fi generalizată nu la două, ci la mai multe relaţii. În fapt, dependenţa de joncţiune corespunde existenţei mai multor legături care sunt independente în cadrul relaţiei şi care vor fi izolate prin descompunerea în 5NF.

Cele trei exemple

• Pe baza dependenţei de joncţiune demonstrate, tabela XYZ din figura 6.14 este transpusă în 5NF prin descompunerea sa în relaţiile XY, XZ şi YZ, prezentate în figura 6.15.

• Relaţia PCS din figura 6.17, care se află în 4NF, este adusă în 5NF sub forma tabelelor PC, PS şi CS din figura 6.18.

• Datorită DJ notate * {CumpărătorMagazin, MagazinBancă, CumpărătorBancă} (figura 6.19), în 5NF relaţia CREDITE_RATE are compoziţia din figura 6.20.

CumpărătorMagazin

Cumpărător	Magazin
Alexa V. Gheorghe	Onitrade Ltd.
Alexa V. Gheorghe	Domo
Georgescu P. Mircea	Onitrade Ltd.
Brava H. Ion	Moldova
Brava H. Ion	Onitrade Ltd.
Fînaru W. Luminiţa	Onitrade Ltd.
Fînaru W. Luminiţa	Domo
Fînaru W. Luminiţa	Moldova

MagazinBancă

Magazin	Bancă
Onitrade Ltd.	BRD
Domo	BCR
Onitrade Ltd.	Banca Ion Ţiriac
Moldova	Banca Ion Ţiriac
Moldova	Banca Albina

CumpărătorBancă

Cumpărător	Bancă
Alexa V. Gheorghe	BRD
Alexa V. Gheorghe	BCR
Georgescu P. Mircea	Banca Ion Ţiriac
Brava H. Ion	Banca Ion Ţiriac
Brava H. Ion	BRD
Fînaru W. Luminiţa	Banca Ion Ţiriac
Fînaru W. Luminiţa	BCR
Fînaru W. Luminiţa	Banca Albina

Figura 6.20. Relaţia CREDITE_RATE în 5NF

6.5. Alte forme normale

Cele cinci forme normale au fost completate şi cu alte tentative de ameliorare a schemei bazei prin identificarea unor restricţii şi tipuri de dependenţă. Ronald Fagin, iniţial, a demonstrat că orice relaţie aflată în 5NF nu mai poate fi descompusă fără pierderi de informaţii[32], însă tot el este autorul a, probabil, celei mai cunoscute forme normale post-5FN - *forma normală domeniu-cheie* (DK/NF - *Domain-Key Normal Form*)[33]. DK/FN nu utilizează dependenţe funcţionale, multivaloare sau de joncţiune. O relaţie este în DK/NF dacă *orice restricţie poate fi determinată doar cunoscând ansamblul numelor atributelor şi domeniilor pe care sunt definite acestea, precum şi cheile relaţiei*[34], altfel spus, *toate restricţiile şi dependenţele care se*

32. [Fagin 79], citat în [Miranda & Busta 90], vol. II, p. 152.
33. [Fagin 81].
34. [Fagin 81].

manifestă într-o relaţie sunt o consecinţă a restricţiilor de domeniu şi de cheie ale relaţiei[35]. După Fagin, o schemă este în DK/NF dacă şi numai dacă nu prezintă anomalii de inserare şi ştergere[36]. Trebuie precizat însă că Fagin priveşte diferit noţiunea de anomalie decât teoria clasică a normalizării (accepţiunea lui Codd): pentru el, spre exemplu, o anomalie de inserare apare atunci când adăugarea aparent corectă a unei linii într-o tabelă violează una dintre restricţiile relaţiei.

Deşi pomenită în destule lucrări, DK/NF a rămas la stadiul de deziderat, neexistând lucrări care să se constituie într-un îndreptar pentru aducerea efectivă a schemei unei baze de date în DK/NF. Probabil, problema principală ţine de dificultatea definirii restricţiilor laborioase ce se manifestă în aplicaţiile complexe actuale.

În afara acestei forme normale, Chris Date mai pomeneşte despre alte două[37]:

- forma normală „selecţie-reuniune" (*„Restriction-union" normal form*), care se referă la fragmentarea unei relaţii (tuplurilor) în relaţii diferite prin *selecţie*, şi nu prin *proiecţie*, cum face normalizarea clasică; de exemplu, în relaţia CLIENŢI_SUA s-ar regăsi clienţii americani, în CLIENŢI_UK cei din Anglia etc., ca să ne oprim doar la celebra axă băsesciană. Relaţia iniţială s-ar reface prin reuniune, nu prin joncţiune.
- a şasea formă normală – 6NF, pe care Date o defineşte ca acea situaţie în care într-o schemă nu există nici o dependenţă de joncţiune netrivială, o dependenţă de joncţiune fiind trivială dacă şi numai dacă cel puţin una dintre proiecţii este definită pe toate atributele relaţiei[38]. După cum observaţi, lucrurile nu stau deloc simplu, aşa că implicaţiile practice asupra proiectării bazelor de date nu se întrezăresc.

35. [Elmasri & Navathe 00], p. 523.
36. [Fagin 81]
37. [Date 04], pp. 399-400.
38. [Date 04], pp. 762-763.

CAPITOLUL 7

Surogate, reguli, incluziuni. Soluţii SQL şi Oracle. Un caz practic

Cu atuurile şi limitele ei, normalizarea reprezintă un excelent ghid pe care l-am folosit pentru a obţine o schemă acceptabilă a bazei de date. Pentru ceea ce urmează încercăm să legăm normalizarea de alte elemente structurale ale bazei, cum ar fi identificarea mai simplă a entităţilor şi restricţiile complexe ale datelor din bază. De asemenea, acest capitol încearcă să demonstreze că, în multe cazuri, în normalizare, pe lângă dependenţele funcţionale, multivaloare şi de joncţiune, esenţiale sunt şi cele ce fac parte dintr-o specie destul de neglijată - dependenţele de incluziune.

De asemenea, vom începe să prezentăm şi modalităţi practice de declarare/ implementare a diferitelor categorii de restricţii discutate, iar cazul practic de la finalul capitolului încearcă să lege elementele teoretice presărate în primele şase capitole, să ilustreze câteva dintre frământări, dacă nu chiar convulsii, care însoţesc procesul de elaborare a schemei bazei de date pentru o aplicaţie reală, chiar în condiţiile în care datele problemei nu sunt complicate din cale-afară.

7.1. Chei surogat

Folosind limbajul de lemn al tehnologiei informaţiei, putem spune că bazele de date stochează persistent şi gestionează varii informaţii privitoare la fenomene, procese, tranzacţii, persoane etc. din mai toate domeniile de activitate, în funcţie de aplicaţiile pentru care sunt proiectate şi implementate. Una dintre problemele de căpătâi în lucrul cu aceste entităţi, procese, tranzacţii etc. ţine de posibilitatea identificării lor fără ambiguitate.

7.1.1. *Nevoia de surogate*

Închipuiţi-vă ce-ar însemna ca firma noastră să vireze câteva sute de milioane de lei în contul unui client, plătind astfel o factură, iar contul respectiv să fie confundat cu un altul, al altui client. Sau ce s-ar întâmpla dacă la plata „pe card" a salariilor s-ar produce niscaiva confuzii: cei mai oropsiţi s-ar oripila văzând cât câştigă şefii (confirmând luminoasa teză „impozita-i-ar naiba ! " a unor preşedinţi retardaţi istoric

şi economic), iar şefii ar fi cel puţin la fel de oripilaţi, dacă nu chiar îngroziţi de mânia populară/proletară.

Aşa că vedem în jurul nostru tot soiul de „ingrediente" menite a face diferenţieri sau, altfel spus, a identifica fiecare persoană, carte, factură, plată etc. Privind retrospectiv, în relaţia STUDENŢI_EXAMENE – figura 2.1, paragraful 2.1 – fiecare student este identificat de matricolul său, iar disciplinele au un cod (`CodDisc`) unic prin care evităm confuzia ce s-ar putea genera prin folosirea exclusivă a denumirii disciplinei (respectiv, a numelui studentului). De asemenea, valoarea atributului `NrFact` este unică pentru orice factură emisă de firma noastră şi trimisă unui client (baza de date VÂNZĂRI). `ISBN`-ul este un cod unic asociat, la nivel mondial, unei cărţi tipărite, în timp ce cota identifică într-o bibliotecă un exemplar al unei cărţi. Codul numeric personal (`CNP`-ul) este cel mai bun mod de a identifica o persoană, în timp ce codul fiscal este un element de încredere când ne propunem să identificăm un client.

Din contră, în paragraful 3.4, exasperaţi de problemele ridicate de nulitatea atributelor cheie din relaţia CODURI_NOI_V2 (figura 3.18), am apelat la o soluţie „gordiană", introducând un atribut cu oarecare doză de artificialitate, `Id`, care ar identifica fiecare grup de adrese atribuite unui cod poştal (figura 3.20, relaţia CODURI_NOI_V3). Atributul `Id` este un exemplu de *cheie surogat*, adică un câmp oarecum artificial de care vorbeam. De altminteri, noţiunea de surogat desemnează un produs artificial sau sintetic care este utilizat drept substitut pentru un produs natural. O cheie surogat este o cheie artificială sau sintetică utilizată ca substitut al unei chei „naturale".

Pe parcursul precedentelor capitole au mai fost strecurate chei surogat precum: `IdFilm` (baza de date FILMOGRAFIE), `CodFamilie` (relaţia DOTARE_JUCĂRII), `CodCl`, `CodPr`, `CodÎnc` şi `CodDoc` (baza de date VÂNZĂRI), `CodProf` (baza de date EXAMENE – paragraful 5.5.2) sau `IdProf` (relaţia PROF_DISCIPLINE din paragraful 5.5.3).

De cele mai multe ori, cheile surogat simplifică graful dependenţelor funcţionale şi, implicit, sarcina demarcării relaţiilor. Astfel, revenind la relaţia LINII_ARTICOLE_CONTABILE din figura 4.1, ştiind că o notă contabilă are mai multe operaţiuni care, la rândul lor, conţin mai mai multe corespondenţe `ContDebitor-ContCreditor`, introducem două atribute care să preia explicaţiile/comentariile despre note, respectiv operaţiuni, `ExplicaţiiNotă` şi `ExplicaţiiOp`. Din paragraful 4.1.2 ştim că, în funcţie de modul de contare, configuraţia dependenţelor poate fi sensibil diferită. Pentru uniformizarea celor două variante de numerotare a notelor contabile, putem recurge la două atribute de tip cheie surogat. `IdNotăContabilă` va fi un număr unic asociat fiecărei note contabile, iar `IdOperaţiune` va identifica orice operaţiune, indiferent de nota contabilă din care face parte. Graful ia configuraţia din figura 7.1.

Schema obţinută prin decuparea dependenţelor din graf este:

NOTE_CONTABILE {`IdNotăContabilă`, `NrNotă`, `Data`, `ExplicaţiiNotă`}
OPERAŢIUNI {`IdOperaţiune`, `NrOp`, `IdNotăContabilă`, `ExplicaţiiOp`}
DETALII_OPERAŢIUNI {`IdOperaţiune`, `ContDebitor`, `ContCreditor`, `Suma`}

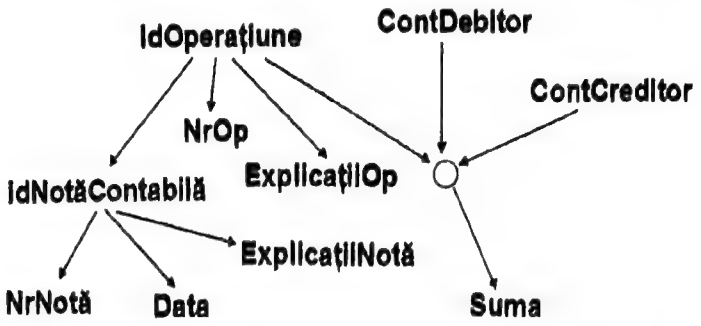


Figura 7.1. Două chei surogat pentru LINII_ARTICOLE_CONTABILE

Un alt caz în care folosirea cheii surogat modifică sensibil lucrurile este cel al relaţiei DOTARE_JUCĂRII_1, reprezentată în figura 3.14 (finalul paragrafului 3.3.3), despre care ultima dată am discutat în paragraful 6.2 - vezi figura 6.13. În condiţiile date la acel moment, această relaţie necesita trei atribute care să confere unicitate fiecărui tuplu - (`CodFamilie`, `Copil`, `Jucărie`). Introducem un atribut suplimentar, `IdJucărie`, nimic altceva decât o cheie surogat prin care diferenţiem jucăriile între ele. Aceasta ar fi a doua cheie surogat din relaţie, după `CodFamilie`. Coroborând dependenţele funcţionale din paragraful 4.1.2 cu dependenţele multivaloare din paragraful 6.1 şi discuţia din 6.2, în condiţiile în care, într-o familie, orice jucărie este cumpărată în N exemplare, unde N reprezintă numărul copiilor din acea familie, grafurile sunt cele din figura 7.2, diferenţierea fiind făcută de momentul cumpărării celor N „instanţe" ale jucăriei într-o familie.

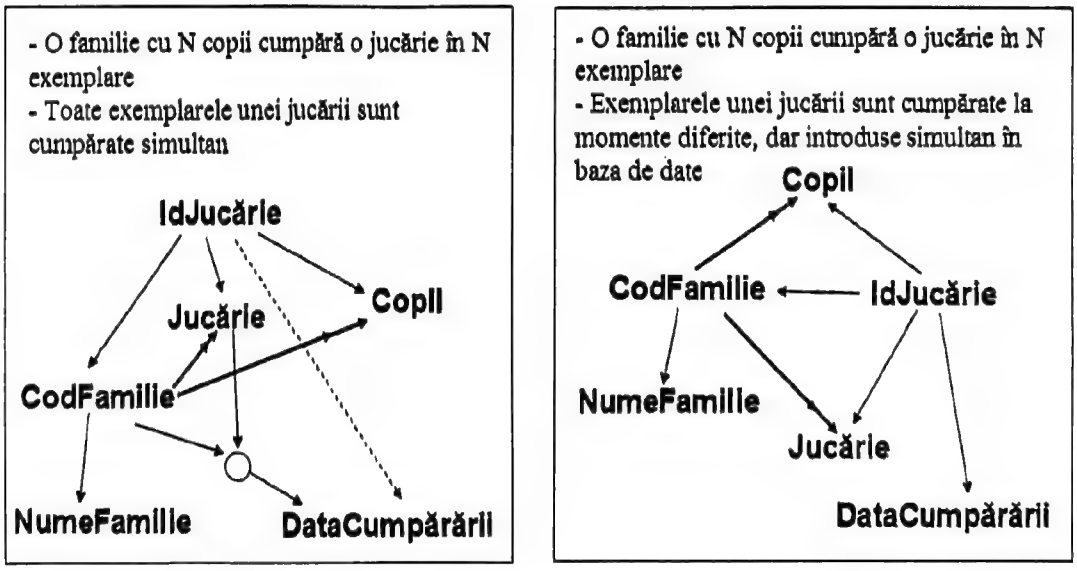


Figura 7.2. Grafurile dependenţelor pentru relaţia DOTARE_JUCĂRII „dotată" cu două chei surogat

Începem cu graful din strânga, cel care reflectă achiziţionarea simultană a câte un exemplar pentru fiecare copil dintr-o familie. Dependenţa funcţională dintre `IdJucărie` şi `DataCumpărării` este una tranzitivă, aşa că s-a recurs la „punctarea" sa, pentru a nu fi luată în considerare. Din dependenţele funcţionale am obţine relaţiile:

FAMILII {`CodFamilie`, `NumeFamilie`},

JUCĂRII_COPII {IdJucărie, Jucărie, CodFamilie, Copil}
şi
JUCĂRII_FAMILII {CodFamilie, Jucărie, DataCumpărării}.

Dacă ţinem cont de dependenţele multivaloare, s-ar mai adăuga acestor relaţii şi :

COPII_FAMILII {CodFamilie, Copil}
şi
JUCĂRII_FAMILII2 {CodFamilie, Jucărie}.

Ambele sunt redundante, însă în timp ce COPII_FAMILII se constituie ca un soi de nomenclator al copiilor dintr-o familie, permiţând preluarea în baza de date şi a copiilor din familiile fără nici o jucărie înregistrată (aşa cum observam la finalul paragrafului 6.2), JUCĂRII_FAMILII2 nu se justifică cu nici un preţ, toate informaţiile furnizate de aceasta găsindu-se în relaţia JUCĂRII_COPII.

Continuăm cu graful din partea dreaptă a figurii 7.2. Faptul că exemplare diferite din aceeaşi jucărie pot fi cumpărate la momente diferite, chiar dacă se preiau simultan, ne scuteşte de dependenţa funcţională cu sursa compusă vizibilă pe graful din stânga. De asemenea, dependenţa funcţională dintre IdJucărie şi DataCumpărării nu mai este tranzitivă. Decuparea DF generează relaţiile :

FAMILII {CodFamilie, NumeFamilie}
şi
JUCĂRII {IdJucărie, Jucărie, CodFamilie, Copil, DataCumpărării}.

Dacă ţinem cont de DMV, am mai obţine relaţiile COPII_FAMILII {CodFamilie, Copil} şi JUCĂRII_FAMILII2 {CodFamilie, Jucărie}. Din raţiunile expuse mai sus, ar fi bine să o păstrăm doar pe prima şi să renunţăm la cea din urmă.

Din moment ce am introdus un atribut pentru identificarea fiecărui exemplar dintr-o jucărie, pare cât se poate de normal să folosim o asemenea cheie surogat pentru identificarea fiecărui copil din baza de date. La drept vorbind, acest atribut există şi nu e câtuşi de puţin surogat – CNPCopil. Noi însă, fideli tematicii acestui paragraf, ne vom preface că nu avem de unde să ştim CNP-urile copiilor, aşa că vom lucra cu IdCopil.

Noul rând de grafuri ale dependenţelor capătă forma din figura 7.3. Motivul principal pentru care ne ocupăm de acest caz este apariţia în ambele grafuri a unei simetrii ciudate între o dependenţă funcţională şi una multivaloare. Astfel, în graful din partea stângă, IdCopil $\longrightarrow$ CodFamilie, dar şi CodFamilie $\longrightarrow\!\!\!\!\rightarrow$ IdCopil. Simetria ce apare în graful din dreapta se manifestă între aceleaşi atribute.

Nu este prima dată în această carte când ridicăm din umeri şi ne întrebăm ce-i de făcut. Primul impuls este de a elimina una dintre dependenţe. Cel mai pieziş ne uităm spre cea multivaloare, însă trebuie să ne temperăm elanul atâta vreme cât o dependenţă multivaloare se stabileşte între două atribute, dar numai în prezenţa unui al treilea : CodFamilie $\longrightarrow\!\!\!\!\rightarrow$ IdCopil|Jucărie. Ne va fi greu să justificăm

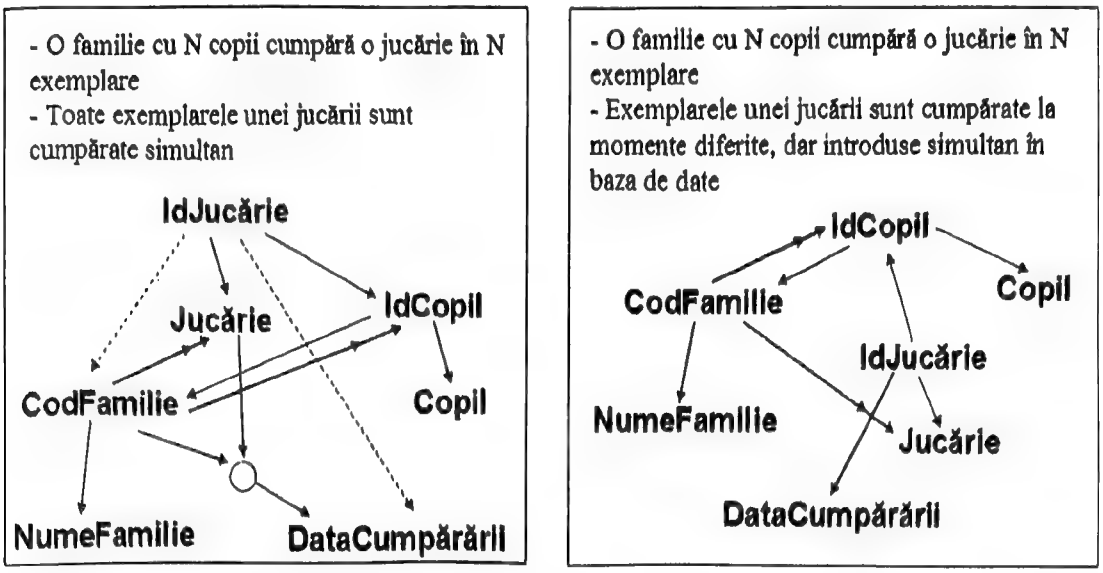


Figura 7.3. Grafurile dependenţelor pentru relaţia DOTARE_JUCĂRII „dotată" cu trei chei surogat

prezenţa în graf a DMV `CodFamilie` $\longrightarrow\!\!\!\!\rightarrow$ `Jucărie`, atâta vreme cât `CodFamilie` $\longrightarrow\!\!\!\!\rightarrow$ `IdCopil` este eliminată din calcul. Aşa că deocamdată nu renunţăm la nimic din simetrie...

Ocupându-ne de graful din stânga imaginii, observăm că acum două dintre dependenţe sunt punctate, pe motiv de tranzitivitate. Pe baza DF obţinem relaţiile:

FAMILII {`CodFamilie`, `NumeFamilie`},
COPII {`IdCopil`, `Copil`, `CodFamilie`},
CUMPĂRĂRI_JUCĂRII {`CodFamilie`, `Jucărie`, `DataCumpărării`}
şi
JUCĂRII_COPII {`IdJucărie`, `Jucărie`, `IdCopil`}.

Din DMV, s-ar mai adăuga:
COPII_FAMILII {`CodFamilie`, `IdCopil`}
şi
JUCĂRII_FAMILII2 {`CodFamilie`, `Jucărie`}.

Informaţia furnizată de COPII_FAMILII, copiii unei familii, este furnizată de COPII, iar la JUCĂRII_FAMILII2 putem renunţa fără remuşcări, atâta vreme cât există CUMPĂRĂRI_JUCĂRII. Iată cum am ajuns ca DMV să nu aibă nici o incidenţă directă în schema finală a bazei de date! Graful din partea dreaptă a figurii rămâne drept temă pentru acasă.

7.1.2. *Puncte de vedere privind cheile surogat*

Părerile despre cheile surogat sunt, firește, împărţite. Există autori care fac distincţia între chei surogat definite/controlate de utilizatori şi chei surogat gestionate exclusiv de sistem (SGBD). Pentru Codd, *cheile primare definite şi controlate de utilizator*,

folosite ca surogate permanente pentru entităţi, ar prezenta trei dificultăţi la întrebuinţare[1]:

- Valorile cheilor surogat trebuie uneori schimbate. De exemplu, atunci când două companii fuzionează, angajaţii trebuie reuniţi într-o sigură tabelă, şi nu este exclus ca, din pricina suprapunerilor, unele mărci să fie modificate.
- Două relaţii ar putea avea chei surogat definite de utilizator pe domenii diferite, chiar dacă entităţile pe care le identifică sunt aceleaşi; de exemplu, o firmă poate fi deopotrivă client şi furnizor al companiei noastre. În tabela FURNIZORI, firma ar putea fi identificată prin `CodFurnizor`, iar în tabela CLIENŢI, prin `CodClient`. Deşi firma este aceeaşi, ea este identificată prin două chei surogat diferite.
- Uneori este necesară preluarea informaţiei despre o entitate înaintea atribuirii de către utilizator a unei chei surogat pentru aceasta sau, pe de altă parte, păstrarea valorii cheii surogat chiar şi după ce entitatea nu mai constituie subiect al aplicaţiei. Cazurile tipice sunt cele ale candidaţilor pentru angajarea pe un post (li se va atribui o marcă doar dacă vor fi angajaţi, atribuirea petrecându-se după angajare) sau cele ale angajaţilor pensionaţi.

Cele trei dificultăţi îi sunt suficiente lui Codd pentru a justifica folosirea *cheilor surogat sistem*, pe care utilizatorul nu le poate nici modifica, iar uneori nici măcar vizualiza. Astfel, fiecare entitate ar avea propria cheie surogat, unică la nivelul întregii baze de date. Două chei surogat sunt identice dacă şi numai dacă desemnează aceeaşi entitate. Codd se gândea chiar şi la o comandă specială de fuzionare (COALESCE) a două chei surogat care desemnează, în fapt, aceeaşi entitate. Mai mult, în articolul său din 1979 publicat în *ACM Transactions on Database Systems*, autorul pledează pentru un model relaţional extins în care, printre multe altele, unul dintre domenii să servească drept sursă pentru toate cheile surogat din bază.

La întrebarea „Ce este de preferat, să folosim cheile primare «naturale» sau pe cele surogat?" nu există un răspuns universal sau, chiar dacă ar exista, am evita să-l dăm. Atunci când atributul/atributele ce identifică entitatea sunt disponibile şi stabile în timp, soluţia „naturală" este mai la îndemână: `CNP`-ul pentru persoane, `ISBN`-ul pentru cărţi, `CodFiscal` pentru firme. Când apar probleme de identificare, iar unicitatea este asigurată de un mare grup de atribute, sau când atributul/atributele potenţiale chei nu prezintă stabilitate pe termen lung, atunci soluţia „surogat" este cea mai indicată. La exemplele din acest paragraf mai putem adăuga:

- Seria şi numărul de buletin/carte de identitate – deşi identifică fiecare persoană, peste un număr de ani, datorită schimbării buletinului/cărţii de identitate, combinaţia îşi schimbă valoarea, iar dacă s-au făcut deja arhivări, vor apărea complicaţii.
- Matricol – în multe şcoli, licee şi universităţi s-a practicat un sistem simplu de atribuire a numărului matricol, astfel încât, după câţiva ani, un matricol era

1. [Codd 79].

„reciclat". Căutarea în baza de date arhivă a liceului ar ridica probleme mari dacă intervalul care interesează este de ordinul deceniilor. Necazuri similare ar apărea dacă unele firme ar recicla, în timp, mărcile angajaților, numerele de inventar ale mijloacelor fixe etc.

Asemenea gen de discuții este valabil însă și la reciclarea cheilor surogat, așadar ideea folosirii unor surogate-sistem se poate dovedi benefică.

Dintre materialele dedicate cheilor surogat, v-am recomanda pe cele scrise de Mike Lonigro[2], Ian Harrington[3], Graeme C. Simsion[4] și Fabian Pascal. În ceea ce mă privește, nu împărtășesc nici lehamitea unor autori față de cheile surogat, mergându-se, în acest sens, până la a le „demasca" drept identificatoare deghizate de obiecte (celebrele OID-uri din orientarea pe obiect), dar nici frenezia „surogării" manifestată de mulți proiectanți de baze de date, deși, recunosc, frenezia cu pricina nu are nimic toxic în ea, ci, mai degrabă, ține de un soi de lene (subscriu cu toată inima la ideea că lenea nu e toxică, dacă nu e însoțită de „aditivi").

O altă temere legată de cheile surogat privește îngreunarea accesului la informațiile din baza de date, întrucât acesta nu se mai realizează prin atribute explicite, ci prin numere destul de irelevante pentru obiectul/entitatea în cauză. Or, utilizatorii obișnuiți ar fi obligați să rețină kilograme întregi de cifre nesuferite, doar pentru a obține datele de care au nevoie. Nici acest neajuns nu trebuie să descurajeze, deoarece trebuie să vedem o aplicație, indiferent de tipologia sa, pe cel puțin două straturi, *date* și *interfață*. În majoritatea copleșitoare a cazurilor, utilizatorii interacționează cu baza doar prin meniuri, rapoarte și mai ales formulare, iar cheile surogat pot fi chiar ascunse.

În orice caz, dacă nu am reușit să tranșăm discuția chei naturale-chei surogat, măcar să arătăm cu degetul pe cele mai toxice: cheile inteligente. Astfel, au existat firme în care angajații compartimentelor *personal-salarizare* audiaseră sau citiseră în cursurile de baze de date sau de analiză/proiectare despre teoria codurilor și, la angajarea unei persoane la compartimentul *Contabilitate*, îi atribuiau o marcă de tip *CTB85CC101*, în care primele trei litere semnalau că este încadrat la compartimentul *Contabilitate*, iar cele două *C*-uri semnalau biroul *Calculația costurilor*, în cadrul compartimentului *Contabilitate*. Partizanii acestui sistem argumentau că marca ar conține, în acest fel, informații prețioase (compartimentul și biroul) despre fiecare angajat, cheia fiind chiar, după unii autori, inteligentă.

După cinci ani însă, persoana respectivă se transferă la compartimentul *Financiar*. Asta înseamnă că marca, fiind inteligentă, trebuie schimbată. Costul schimbării depinde de întrebarea: unde se află cele 5 (ani) * 12 (luni) înregistrări despre calculul lunar al salariului, plus cele 5(ani) * 12 (luni) * 20 (zile lucrătoare) înregistrări, dacă pontajul se preia zilnic? Răspunsul poate fi însoțit de apăsătoare dureri de cap.

2. [Lonigro 98].
3. [Harrington 02], pp. 77-82.
4. [Simsion 01], pp. 282-285.

7.1.3. *Declararea/obţinerea cheilor surogat*

Modalităţile prin care un atribut poate fi declarat cheie surogat diferă de la SGBD la SGBD. Trei dintre variante sunt folosite ceva mai des. Prima este declararea unui atribut de tip *autoincrement*. Spre exemplu, în PostgreSQL există tipul de dată SERIAL:

```
CREATE TABLE note_contabile (
  idnotacontabila SERIAL PRIMARY KEY
...
)
```

În momentul inserării unei linii noi în tabelă, atributul `IdNotăContabilă` va primi, pe rând, valorile: 1, 2, ... până la 2147483647[5]. Dacă se doreşte crearea unei chei surogat de tip sistem, iar valoarea maximă de mai sus nu este de ajuns, se poate folosi tipul BIGSERIAL, cu o plajă imensă.

Şi în Visual FoxPro, o dată cu versiunea 8, un atribut poate fi declarat cheie surogat folosind clauza `AUTOINC`:

```
CREATE TABLE note_contabile (;
idnotacont INTEGER AUTOINC NEXTVALUE 1001 STEP 1 PRIMARY KEY;
)
```

Clauza adiţională `NEXTVALUE` este utilă atunci când se doreşte ca valoarea iniţială să nu fie 1, iar `STEP` stabileşte mărimea incrementată. În versiunile mai vechi ale VFP era necesară folosirea clauzei `DEFAULT`, care făcea apel la o funcţie stocată[6]:

```
CREATE TABLE note_contabile (;
idnotacont NUMBER(10) DEFAULT vi_nc_idnota() PRIMARY KEY;
)
```

funcţia fiind cea din *listing*-ul 7.1.

O altă soluţie, disponibilă în Oracle şi PostgreSQL, presupune folosirea secvenţelor, care sunt obiecte ale bazei de date ce furnizează la fiecare invocare a clauzei `NextVal` o valoare unică:

```
CREATE SEQUENCE seq_idnota START WITH 1001
 MINVALUE 1001 MAXVALUE 9999999999 NOCACHE NOCYCLE ORDER
```

Secvenţa `seq_idnota` creată va furniza valori ordonate strict după momentul apelului, cuprinse între 1001 şi 9999999999, iar după atingerea limitei superioare

5. Vezi documentaţia PostgreSQL de pe *www.postgresql.org*.
6. Vezi şi [Fotache ş.a. 02], pp. 184-188.

Listing 7.1. Funcţia stocată care furnizează valoarea implicită a cheii surogat

```
PROCEDURE vl_nc_idnota
LOCAL v_idnota (1,1)
v_idnota=0
SELECT MAX(idnotacont) FROM note_contabile ;
    INTO ARRAY v_idnota
IF v_idnota (1,1) = 0 && e prima nota contabila
    RETURN 1001
ELSE
    RETURN v_idnota(1,1) + 1
ENDIF
ENDPROC
```

secvenţa se blochează (clauza NOCYCLE). În lipsa clauzei NOCYCLE, după atingerea valorii maxime, următoarea valoare furnizată este cea minimă. Secvenţa va fi folosită în declanşatorul de inserare a tabelei NOTE_CONTABILE. Scriptul de creare a celor trei tabele discutate în paragraful 7.1.1 (vezi figura 7.1) constituie subiectul *listing*-ului 7.2, care poate fi descărcat de la adresa *http : //www.feaa.uaic.ro/cercetare/publicatii*). Iată corpul declanşatorului din *listing*-ul 7.3[7].

Listing 7.3. Declanşatorul Oracle pentru valoarea implicită a cheii surogat

```
CREATE OR REPLACE TRIGGER trg_note_contabile_ins
    BEFORE INSERT ON note_contabile FOR EACH ROW
BEGIN
    SELECT seq_idnota.NextVal INTO :NEW.IdNotaContabila FROM dual ;
END ;
```

7.2. Restricţii simple şi complexe. Atribute redundante ajutătoare

Nu întotdeauna atributele implicate în schema bazei pot fi identificate cu uşurinţă. Mai mult, deseori nu strică o doză rezonabilă de artificial, de improvizaţie, doză care poate salva pe termen lung o schemă sau măcar să o scutească de multe necazuri. Deşi mai puţin intuitiv decât modelul obiectual, relaţionalul are certe atuuri în ceea ce priveşte mecanismul de declarare a restricţiilor şi mai ales în ceea ce priveşte opţiunile de extragere a informaţiilor din bază (mecanismul de *manipulare a datelor*). Chiar şi aşa, rămân în discuţie o mare parte dintre restricţiile care ţin de integritatea bazei de date, dar sunt legate nu atât de reguli intrinseci modelului (cheie primară, restricţie de entitate, integritate referenţială), cât de regulile aplicaţiei sau,

7. Detalii privind declanşatoarele pentru valori implicite în Oracle sunt prezentate în paragraful 11.2 al lucrării [Fotache ş.a. 03], pp. 386-391.

altfel spus, reguli ale afacerii. Din păcate, nici un model de date nu oferă un mecanism infailibil de declarare a tuturor restricţiilor semantice dintr-o bază de date.

7.2.1. *Reguli la nivel de atribut şi înregistrare*

Până în acest paragraf, cele mai pomenite restricţii semantice ale bazei au fost dependenţele funcţionale, multivaloare şi de joncţiune. Probabil însă că practicienii sunt cel mai bine familiarizaţi cu altfel de restricţii: reguli de validare la nivel de atribut (*field validation rule*) şi reguli de validare la nivel de înregistrare (*record validation rule*).

Să luăm în discuţie schema bazei de date EXAMENE din paragraful 5.5.2 (cazul practic nr. 2), al cărei graf DF constituie subiectul figurii 5.17. Reamintim că baza de date preia informaţii legate de cursurile predate şi rezultatele obţinute la examinările organizate la Facultatea de Economie şi Administrarea Afacerilor. Chiar şi la o privire sumară a celor şase relaţii, se pot desprinde câteva *reguli de validare la nivel de atribut*:

- în relaţia STUDENŢI:
  - `An` (anul de studiu al studentului) este un număr natural cuprins între 1 şi 5;
  - `Modul` – modulul în care poate fi înscris un student de la FEAA – este 1 sau 2;
  - `Spec` (specializarea) poate avea doar una dintre cele 10 valori: *Finanţe-Bănci*, *Contabilitate*, *Informatică economică*, *Marketing* etc.;
- în relaţia DISCIPLINE, numărul de credite ale unei discipline (`NrCredDisc`) nu poate fi mai mare de 8;
- în PROFESORI, atributul `GradProf` poate avea doar una dintre cele şase valori: *colaborator*, *preparator*, *asistent*, *lector*, *conferenţiar* şi *profesor*;
- examenele sunt programate doar în lunile ianuarie-februarie şi mai-iunie-iulie, corespunzătoare celor două sesiuni; prin urmare, componenta *lună* a atributului `DataEx` în tabelele SĂLI_EX şi NOTE trebuie să se încadreze în cele cinci valori;
- nota maximă obţinută este 10, iar pentru *absent* valoarea convenită a notei este zero (discutabil!).

Acest gen de restricţii poate fi legat în modelul relaţional de noţiunea de domeniu de valori. După cum vom discuta spre finalul acestei lucrări, majoritatea SGBD-urilor nu se bazează pe definirea explicită a domeniului, ci prin opţiuni de declarare a acestor reguli de o manieră intuitivă, prin comenzi SQL sau interactiv, de către fiecare SGBD.

În privinţa *regulilor la nivel de înregistrare*, principala diferenţă faţă de cele la nivel de atribut ţine de numărul atributelor implicate simultan, cel puţin două. De exemplu, în relaţia STUDENŢI, ţinând seama că primii doi ani de studiu sunt comuni, iar înscrierea la o specializare se face de către fiecare student începând cu anul al III-lea, este evident că pentru fiecare linie în care `An` este 1 sau 2, valoarea atributului `Spec` nu trebuie să se regăsească în lista celor 10 descrisă mai sus; aşadar, avem nevoie şi de o valoare de genul *Trunchi comun* care se adaugă domeniului de valori pentru `Spec`; firește, am putea recurge şi la valoarea NULL, însă fideli principiului *ocoliţi NULL-ul ori de câte ori aveţi ocazia*, vom marşa pe varianta trunchiului comun.

Încercând să ne apropiem mai mult de o formalizare aproximativă, putem redacta restricţia la modul: dacă valoarea atributului `An` este 1 sau 2, atunci obligatoriu atributul `Spec` ia valoarea *Trunchi comun*!

7.2.2. *Reguli, atribute redundante şi declanşatoare*

Cele două tipuri de restricţii acoperă o zonă destul de subţire din clasa regulilor ce pot guverna o aplicaţie/bază de date. Încercăm să trecem într-un registru mai dificil. Ne întoarcem la baza de date BIBLIOTECĂ, a cărei schemă a fost finalizată în paragraful 6.2 (vezi figura 6.10) prin aducerea în 4NF. Din raţiuni bugetare, conducerea facultăţii/bibliotecii stabileşte următoarea restricţie: *o carte nu poate fi achiziţionată în mai mult de 10 exemplare!* Cum se poate implementa o asemenea restricţie? Norocul nostru este că, între cele cinci relaţii ale bazei de date, e uşor de identificat cea împricinată - EXEMPLARE {`Cotă`, `ISBN`}. Respectarea acestei restricţii presupune ca, la inserarea unei linii în această tabelă sau la modificarea unui ISBN, să se verifice dacă numărul cotelor pentru „noul" ISBN (valoarea atributului `ISBN` de pe linia inserată sau modificată) nu depăşeşte 10, situaţie în care inserarea/modificarea să fie blocată.

Acest gen de restricţie poate fi implementată în două moduri. Fără a schimba schema bazei de date, se recurge la declanşatoarele de inserare şi modificare ale tabelei EXEMPLARE, declanşatoare care operează după o logică simplă, aşa cum este prezentat în *listing*-ul 7.5 (*listing*-ul 7.4 poate fi descărcat de pe pagina Web amintită mai sus, conţinând scriptul de creare a tabelelor):

Listing 7.5. Declanşator Oracle de inserare[8] pentru implementarea restricţiei legate de numărul de exemplare dintr-o carte

```
CREATE OR REPLACE TRIGGER trg_exemplare_ins
   BEFORE INSERT ON exemplare
   FOR EACH ROW
DECLARE
   v_nrexemplare NUMBER(5) := 0 ;
BEGIN
   SELECT COUNT(*) INTO v_nrexemplare
   FROM exemplare WHERE isbn = :NEW.isbn ;
   IF v_nrexemplare > 9 THEN
      RAISE_APPLICATION_ERROR (-20801, 'Nr. exemplarelor acestei carti creste peste 10!');
   END IF ;
END ;
```

Variantei i se poate reproşa cel puţin faptul că un mare număr de cărţi poate implica timpi de execuţie destul de greu de trecut cu vederea, deoarece `SELECT COUNT(*)` parcurge întreaga tabelă. Aşa că ne-am putea gândi la o variantă ce pare,

8. Declanşatorul de modificare trebuie să fie redactat astfel încât să preîntâmpine situaţiile de „mutanţă" a tabelei EXEMPLARE (vezi [Fotache ş.a. 03], pp. 396-402).

la prima vedere, mai laborioasă: în tabela TITLURI introducem atributul `NrExemplare`, atribut care va fi actualizat automat prin cele trei declanşatoare ale relaţiei EXEMPLARE: TITLURI {`ISBN, Titlu, Editura, AnApariţie, NrExemplare`}:

```
ALTER TABLE titluri ADD (NrExemplare  NUMBER(4));
```

În continuare, creăm o funcţie `f_NrExemplare`, căreia i se pasează un ISBN şi care returnează numărul de exemplare ale acelui ISBN – vezi *listing*-ul 7.6.

Listing 7.6. Funcţie Oracle ce returnează numărul de exemplare ale unei cărţi

```
CREATE OR REPLACE FUNCTION f_NrExemplare ( isbn_ titluri.isbn%TYPE)
   RETURN NUMBER
IS
   v_nr NUMBER(4) := 0 ;
BEGIN
   SELECT nrexemplare INTO v_nr FROM titluri WHERE isbn=isbn_ ;
   RETURN v_nr ;
END ;
```

Declanşatoarele tabelei EXEMPLARE, care folosesc funcţia, se prezintă astfel:

- Pentru inserare, vezi *listing*-ul 7.7:

Listing 7.7. Versiunea a doua a declanşatorului din *listing*-ul 7.5

```
CREATE OR REPLACE TRIGGER trg_exemplare_ins
 BEFORE INSERT ON exemplare
 FOR EACH ROW
BEGIN
   IF f_NrExemplare (:NEW.isbn) > 3 THEN
      RAISE_APPLICATION_ERROR (-20801,
         'Nr. exemplarelor acestei carti creste peste 10!') ;
   ELSE
      /* se incrementeaza numarul exemplarelor pentru ISBN-ul din linia inserata */
      UPDATE titluri SET NrExemplare = NrExemplare + 1   WHERE isbn = :NEW.isbn ;
   END IF ;
END ;
```

- Pentru modificare, vezi *listing*-ul 7.8:

Listing 7.8. Declanşatorul de modificare a valorii atributului EXEMPLARE.isbn

```
CREATE OR REPLACE TRIGGER trg_exemplare_upd
   BEFORE UPDATE OF isbn ON exemplare
   FOR EACH ROW
BEGIN
   IF f_NrExemplare (:NEW.isbn) > 3 THEN
      RAISE_APPLICATION_ERROR (-20801,
```

```
            'Nr. exemplarelor acestei carti creste peste 10!') ;
    ELSE
        /* se scade cu 1 numarul exemplarelor pentru
            vechiul ISBN (valoarea dinaintea modificarii) */
        UPDATE titluri SET NrExemplare = NrExemplare - 1   WHERE isbn = :OLD.isbn;

        /* se incrementeaza numarul exemplarelor pentru  ISBN-ul din linia inserata */
        UPDATE titluri SET NrExemplare = NrExemplare + 1   WHERE isbn = :NEW.isbn ;
    END IF ;
END ;
```

Un avantaj colateral al acestei variante de declanşator, avantaj deloc neglijabil pentru „oraclişti", este că se elimină problema „mutanţei" tabelei EXEMPLARE.

– Pentru ştergere, vezi *listing*-ul 7.9:

Listing 7.9. Declanşatorul de ştergere a unei linii din tabela EXEMPLARE

```
CREATE OR REPLACE TRIGGER trg_exemplare_del
    BEFORE DELETE ON exemplare FOR EACH ROW
BEGIN
    /* se scade cu 1 numarul exemplarelor pentru ISBN-ul sters */
    UPDATE titluri SET NrExemplare = NrExemplare - 1   WHERE isbn = :OLD.isbn;
END ;
```

Puţini, probabil, vor fi înclinaţi să accepte că această ultimă soluţie ar fi mai indicată decât prima, câtă vreme declanşatoarele de inserare/modificare folosesc o funcţie care execută fraza `SELECT COUNT...`, apoi, ele însele conţin comenzi `UPDATE`. Ei bine, o dată pusă în aplicare, lucrurile nu sunt aşa de negre. Mai întâi, `SELECT`-ul din funcţia `f_NrExemplare` are toate şansele să se execute destul de rapid, întrucât în EXEMPLARE atributul `ISBN` este cheie străină (atributul-părinte fiind `ISBN` din TITLURI), iar la declararea unei asemenea restricţii referenţiale, orice server de baze de date creează un index. Acest index creşte sensibil viteza atât în cazul interogării `SELECT COUNT...`, cât şi la `UPDATE`-uri. În al doilea rând, frecvenţa modificării unui ISBN în EXEMPLARE, ca şi cea a ştergerii unei linii în aceeaşi tabelă, este foarte scăzută, astfel încât, statistic privind lucrurile, declanşatorul cel mai important este cel de inserare.

Marele merit al acestei ultime soluţii este că o pregăteşte pe a treia, care e mult mai simplă. Folosim atributul `NrExemplare` în TITLURI, atribut modificat prin cele trei declanşatoare ale tabelei EXEMPLARE, însă restricţia o implementăm în bază printr-o regulă de validare declarată pentru acest atribut, adică, după sintaxa SQL:

```
ALTER TABLE titluri ADD CONSTRAINT ck_nrexemplare
CHECK (NrExemplare <= 10);
```

Acum, cele trei declanşatoare ale tabelei EXEMPLARE se simplifică sensibil, după cum se observă şi în *listing*-ul 7.10.

Listing 7.10. Noile declanşatoare ale tabelei EXEMPLARE

```
CREATE OR REPLACE TRIGGER trg_exemplare_ins
 BEFORE INSERT ON exemplare FOR EACH ROW
BEGIN
        /* se incrementeaza numarul exemplarelor pentru ISBN-ul din linia inserata */
 UPDATE titluri SET NrExemplare = NrExemplare + 1
        WHERE isbn = :NEW.isbn ;
END ;
/

CREATE OR REPLACE TRIGGER trg_exemplare_upd
    BEFORE UPDATE OF isbn ON exemplare    FOR EACH ROW
BEGIN
    /* se scade cu 1 numarul exemplarelor pentru vechiul ISBN (valoarea dinaintea modificarii) */
    UPDATE titluri SET NrExemplare = NrExemplare - 1 WHERE isbn = :OLD.isbn ;

    /* se incrementeaza numarul exemplarelor pentru ISBN-ul din linia inserata */
    UPDATE titluri SET NrExemplare = NrExemplare + 1WHERE isbn = :NEW.isbn ;
END ;
/

CREATE OR REPLACE TRIGGER trg_exemplare_del
    BEFORE DELETE ON exemplare FOR EACH ROW
BEGIN
    /* se scade cu 1 numarul exemplarelor pentru ISBN-ul sters */
    UPDATE titluri SET NrExemplare = NrExemplare - 1 WHERE isbn = :OLD.isbn ;
END ;
```

Fireşte, nici de funcţie nu mai avem nevoie în noile condiţii, iar soluţia are mult mai multe şanse de a fi acceptată.

Atenţie, însă! Atunci când utilizăm un asemenea atribut redundant, trebuie să fim siguri că modul său de actualizare este bine pus la punct, astfel încât să nu existe nici o modalitate prin care valoarea sa să fie neconformă cu realitatea din baza de date. Or, cel mai sigur mecanism este cel al declanşatoarelor. Din păcate, mulţi dezvoltatori folosesc asemenea artificii doar în interfaţă (formulare) sau logica aplicaţiei, situaţii în care actualizarea atributelor calculate poate fi ocolită, cu sau fără bună ştiinţă.

Un caz şi mai interesant

Să luăm în discuţie şi un alt exemplu. În primul paragraf din acest capitol am modificat baza de date dedicată notelor contabile introducând chei surogat, şi, pe baza grafului dependenţelor din figura 7.1, am ajuns la o schemă alcătuită din trei relaţii: NOTE_CONTABILE {IdNotăContabilă, NrNota, Data, ExplicaţiiNotă}, OPERAŢIUNI {IdOperaţiune, NrOp, IdNotăContabilă, ExplicaţiiOp} şi DETALII_OPERAŢIUNI {IdOperaţiune, ContDebitor, ContCreditor, Suma}. Conţinutul acestor relaţii, obţinut prin prelucrarea relaţiei LINII_ARTICOLE_CONTABILE (figura 4.1), poate fi înţeles din cele câteva înregistrări din figura 7.4 (*listing*-ul de populare a celor trei tabele – 7.11 – este disponibil pe pagina Web amintită).

NOTE_CONTABILE

IdNotaContabilă	NrNota	Data	ExplicaţiiNotă
1001	5	30.11.2004	...
1002	6	01.12.2004	...

OPERAŢIUNI

IdOperaţiune	NrOp	IdNotaContabilă	ExplicaţiiOp
10001	1	5	...
10002	2	5	...
10003	1	6	...

DETALII_OPERAŢIUNI

IdOperaţiune	ContDebitor	ContCreditor	Suma
10001	300	401	10000000
10001	4426	401	1900000
10002	401	5311	7000000
10002	401	5121	4800000
10003	5121	700	5250000

Figura 7.4. Câteva linii din relaţiile bazei de date CONTABILITATE

O restricţie extrem de importantă în contabilitate este că în orice înregistrare contabilă compusă *nu pot fi simultan două sau mai multe conturi şi pe debit, şi pe credit*. Altfel spus, într-o operaţiune contabilă există fie un cont debitor şi unul sau mai multe conturi creditoare, fie un singur cont creditor şi unul sau mai multe debitoare!

Soluţia de început a acestei probleme ţine de valorificarea declanşatoarelor de inserare şi modificare. La fiecare inserare în DETALII_OPERAŢIUNI, declanşatorul trebuie să facă o verificare de genul celei din *listing*-ul 7.12.

Listing 7.12. Verificarea numărului de conturi pe debit şi pe credit la inserare

```
CREATE OR REPLACE TRIGGER trg_do_ins
   BEFORE INSERT ON detalii_operatiuni FOR EACH ROW
DECLARE
   v_nrcontdebit NUMBER(4) := 0 ;
   v_nrcontcredit NUMBER(4) := 0 ;
BEGIN
   SELECT COUNT(DISTINCT ContDebitor), COUNT (DISTINCT ContCreditor)
   INTO v_nrcontdebit, v_nrcontcredit
   FROM
      (SELECT ContDebitor, ContCreditor
         FROM detalii_operatiuni
         WHERE idoperatiune = :NEW.idoperatiune
         UNION
          SELECT :NEW.ContDebitor, :NEW.ContCreditor
          FROM dual
          ) T ;

   IF v_nrcontdebit > 1 AND v_nrcontcredit > 1 THEN
      RAISE_APPLICATION_ERROR (-20852,
```

```
            'Nu pot fi simultan mai multe conturi si pe debit, si pe credit') ;
    END IF ;
END ;
```

A fost nevoie de un mic artificiu pentru că SELECT-ul nu ar fi luat în calcul înregistrarea ce tocmai se adaugă în tabelă (înregistrare care, în fond, determină lansarea declanşatorului) : cele două conturi din noua înregistrare sunt „lipite" celor deja introduse folosind reuniunea (şi tabela DUAL).

Nici la modificarea aceleiaşi tabele lucrurile nu sunt mai tihnite. Pentru a evita problema mutanţei şi pornind de la premisa că printr-un UPDATE nu se modifică niciodată simultan conturile şi identificatoarele mai multor operaţiuni, soluţia Oracle recurge la triada : variabilă publică (pachet), declanşator de actualizare la nivel de linie, declanşator de actualizare la nivel de comandă – vezi *listing*-ul 7.13.

Listing 7.13. Verificarea numărului de conturi pe debit şi pe credit la modificare

```
CREATE OR REPLACE PACKAGE pac_trg IS
    v_idoper operatiuni.idoperatiune%TYPE ;
END ;
/

CREATE OR REPLACE TRIGGER trg_do_upd1
    BEFORE UPDATE OF IdOperatiune, ContDebitor, ContCreditor
    ON detalii_operatiuni FOR EACH ROW
BEGIN
    pac_trg.v_idoper := :NEW.idoperatiune ;
END ;
/

CREATE OR REPLACE TRIGGER trg_do_upd2
    AFTER UPDATE OF IdOperatiune, ContDebitor, ContCreditor
    ON detalii_operatiuni
DECLARE
    v_nrcontdebit NUMBER(4) := 0 ;
    v_nrcontcredit NUMBER(4) := 0 ;
BEGIN
    SELECT COUNT(DISTINCT ContDebitor), COUNT (DISTINCT ContCreditor)
    INTO v_nrcontdebit, v_nrcontcredit FROM detalii_operatiuni
    WHERE idoperatiune = pac_trg.v_idoper ;

    IF v_nrcontdebit > 1 AND v_nrcontcredit > 1 THEN
        RAISE_APPLICATION_ERROR (-20852,
            'Nu pot fi simultan mai multe conturi si pe debit, si pe credit') ;
    END IF ;
END ;
/
```

Nu foarte multe SGBD-uri sunt dispuse să ofere suport pentru asemenea gen de artificii, aşa că ne bate gândul să recurgem la unul sau mai multe atribute redundante

care să mai limpezească din ape. Aşadar, introducem în OPERAŢIUNI două atribute, numite `NrConturiDebitoare` şi `NrConturiCreditoare`, care să furnizeze numărul conturilor din debitul şi din creditul fiecărei operaţiuni contabile. Astfel, relaţia ar avea structura: OPERAŢIUNI (`IdOperaţiune`, `IdNotăContabilă`, `ExplicaţiiOp`, `NrConturiDebitoare`, `NrConturiCreditoare`). Curând ne dăm seama că avem o problemă, întrucât cele două atribute trebuie să ia în calcul fiecare cont o singură dată. Să fim mai expliciţi. Prima operaţiune din nota 101, cea care are identificatorul 10001, se scrie în „limbaj" contabil astfel:

%	401		11800000
300		10000000	
4426		1800000	

Operaţiunii îi corespund primele două linii din DETALII_OPERAŢIUNI, deci stocarea se face transformând înregistrarea după următorul calapod:

Debit	Credit	Suma
300	401	10000000
4426	401	1800000

Acest mod de stocare a înregistrărilor este fundamental pentru obţinerea documentelor contabile ce prelucrează notele contabile: fişe de cont, cartea-mare, balanţe de verificare. Adăugând cele două atribute în OPERAŢIUNI, pe prima linie a acestei tabele (corespunzătoare înregistrării contabile pe care o discutăm) valoarea atributului `NrConturiDebitoare` este 2, iar cea a `NrConturiCreditoare` este 1. Declanşatorul de inserare a tabelei DETALII_OPERAŢIUNI nu trebuie să incrementeze pur şi simplu cele două atribute, ci, în mod normal, numai unul, deoarece celălalt se repetă. Bine, dar cum ştie declanşatorul care cont se repetă, pe debit sau pe credit, fără a face SELECT-ul acela impresionant?

Îngroşând gluma, mai introducem două conturi, `UnContDebitor` şi `UnContCreditor`, care vor avea valoarea iniţială (implicită) NULL:

```
ALTER TABLE operaţiuni ADD NrConturiDebitoare NUMBER (3) ;
ALTER TABLE operaţiuni ADD NrConturiCreditoare NUMBER (3) ;
ALTER TABLE operaţiuni ADD UnContDebitor VARCHAR(14) ;
ALTER TABLE operaţiuni ADD UnContCreditor VARCHAR(14) ;
```

Valoarea implicită a acestor patru atribute va fi NULL. La prima introducere a unei corespondenţe contabile pentru o operaţiune, adică la prima inserare a unei linii în DETALII_OPERAŢIUNI, `NrConturiDebitoare` şi `NrConturiCreditoare` vor primi valoarea 1, `UnContDebitor` va lua valoarea `:NEW.ContDebitor` (valoarea de pe linia inserată a atributului `ContDebitor`), iar `UnContCreditor` pe `:NEW.ContCreditor`. La a doua inserare a unei corespondenţe *cont debitor-cont creditor* pentru aceeaşi notă contabilă, atributul `NrConturiDebitoare` trebuie incrementat cu 1 numai dacă valoarea `:NEW.ContDebitor` este diferită de cea a

`UnContDebitor`, iar `NrConturiCreditoare` trebuie incrementat cu 1 doar dacă valoarea `:NEW.ContCreditor` este diferită de cea a atributului `UnContCreditor` (să nu spuneţi că n-aţi înţeles!).

Cu un pic de imaginaţie, reducem întregul declanşator pentru inserare în tabela DETALII_OPERAŢIUNI la o singură, dar generoasă comandă UPDATE – vezi *listing*-ul 7.14.

Listing 7.14. Noul declanşator de inserare ce actualizează atributele „redundante"

```
CREATE OR REPLACE TRIGGER trg_do_ins
   AFTER INSERT ON detalii_operatiuni FOR EACH ROW
BEGIN
   UPDATE operatiuni
   SET  nrconturidebitoare =  NVL(nrconturidebitoare, 1) +
      CASE WHEN :NEW.contdebitor <> NVL(uncontdebitor,:NEW.contdebitor)
      THEN 1 ELSE 0 END,
      nrconturicreditoare =  NVL(nrconturicreditoare, 1) + CASE
      WHEN :NEW.contcreditor <> NVL(uncontcreditor,:NEW.contcreditor)
           THEN 1 ELSE 0 END,
      uncontdebitor = :NEW.contdebitor,
      uncontcreditor = :NEW.contcreditor
  WHERE idoperatiune = :NEW.idoperatiune ;
END ;
```

Acum putem declara liniştiţi restricţia sub forma regulii de validare:

```
ALTER TABLE operaţiuni ADD CONSTRAINT ck_nrconturi
     CHECK ( NOT (NVL(nrconturidebitoare,0) > 1 AND
          NVL(nrconturicreditoare,0) > 1) ) ;
```

În noile condiţii, şi operaţiunile derulate la modificări survenite în tabela DETALII_OPERAŢIUNI se simplifică sensibil. Putem renunţa la pachet şi unul dintre declanşatoare, iar noua formă a *trigger*-ului de modificare este cea din *listing*-ul 7.15.

Listing 7.15. Noul declanşator de modificare în DETALII_OPERAŢIUNI

```
DROP TRIGGER trg_do_upd1 ;
DROP TRIGGER trg_do_upd2 ;
DROP PACKAGE pac_trg ;
/

CREATE OR REPLACE TRIGGER trg_do_upd
   AFTER UPDATE OF IdOperatiune, ContDebitor, ContCreditor
   ON detalii_operatiuni FOR EACH ROW
BEGIN
   -- mai intii, se fac modificarile pentru vechile valori
   UPDATE operatiuni
   SET  nrconturidebitoare = nrconturidebitoare -
```

```
        CASE WHEN nrconturidebitoare > 1 THEN 1   ELSE 0 END,
        nrconturicreditoare =  nrconturicreditoare -
        CASE WHEN nrconturicreditoare > 1 THEN 1 ELSE 0 END
    WHERE idoperatiune = :OLD.idoperatiune ;

    -- apoi, se fac modificarile pentru noile valori
    UPDATE operatiuni
    SET  nrconturidebitoare =  nrconturidebitoare +
        CASE WHEN :NEW.contdebitor <> uncontdebitor THEN 1    ELSE 0 END,
            nrconturicreditoare =  nrconturicreditoare +
        CASE WHEN :NEW.contcreditor <> uncontcreditor THEN 1 ELSE 0 END,
            uncontdebitor = :NEW.contdebitor,
            uncontcreditor = :NEW.contcreditor
    WHERE idoperatiune = :NEW.idoperatiune ;
END ;
```

Listing-ul 7.16 conţine declanşatorul de ştergere care asigură decrementarea numărului de conturi debitoare şi a celui de conturi creditoare şi, în caz că se şterge ultima linie dintr-o operaţiune notabilă, NULL-izarea atributelor `UnContDebitor` şi `UnContCreditor`.

Listing 7.16. Declanşatorul de ştergere în DETALII_OPERAŢIUNI

```
CREATE OR REPLACE TRIGGER trg_do_del
   BEFORE DELETE ON detalii_operatiuni FOR EACH ROW
BEGIN
UPDATE operatiuni
   SET  nrconturidebitoare = nrconturidebitoare -  CASE WHEN nrconturidebitoare > 1
          OR nrconturidebitoare+nrconturicreditoare=2 THEN 1 ELSE 0 END,
           nrconturicreditoare =  nrconturicreditoare - CASE
           WHEN nrconturicreditoare > 1
          OR nrconturidebitoare+nrconturicreditoare=2 THEN 1 ELSE 0 END,
           uncontdebitor = CASE WHEN nrconturidebitoare+nrconturicreditoare=2
             THEN NULL ELSE uncontdebitor END,
           uncontcreditor = CASE WHEN nrconturidebitoare+nrconturicreditoare=2
             THEN NULL ELSE uncontcreditor END
   WHERE idoperatiune = :OLD.idoperatiune ;
END ;
```

7.2.3. *Restricţii şi relaţii noi*

În paragraful 7.2.1 am enumerat câteva posibile reguli de validare la nivel de atribut, printre care şi una pentru relaţia STUDENŢI prin care, în primii doi ani, studenţii urmează un trunchi comun de discipline, iar începând cu anul 3 de studiu îşi aleg specializarea, astfel încât atributul `Spec` trebuie să ia o valoarea pe domeniul: (*Contabilitate şi informatică de gestiune*, *Economie generală*, *Finanţe-Bănci* etc.). Chiar dacă priveşte atributul `Spec`, declararea acestei restricţii printr-o clauză `CHECK` este destul de anevoioasă:

```
CREATE TABLE studenţi (
  matricol ...
  ...
  Spec VARCHAR(50) NOT NULL CHECK
  (Spec IN 'Contabilitate şi informatică de gestiune', '',
   'Informatică economică', ...)
    ...
  );
```

Lista celor 10 specializări, plus *Truchi comun* este una destul de mare, iar dacă se ia decizia implementării aplicaţiei la nivelul unei universităţi precum „Al.I. Cuza" din Iaşi, ale cărei facultăţi patronează zeci de specializări, scrierea clauzei CHECK devine o sarcină pe care o s-o execute doar cel care nu va avea cui să i-o paseze. Ceea ce vă propunem este o soluţie cunoscută multor practicieni: constituirea unei relaţii dedicate specializărilor (SPECIALIZĂRI), urmând ca regula să fie ulterior implementată printr-o restricţie referenţială STUDENŢI_SPECIALIZĂRI.

Dacă dorim să păstrăm acelaşi număr de atribute, creăm o tabelă cu un singur atribut:

```
CREATE TABLE specializări (
spec VARCHAR(50) NOT NULL PRIMARY KEY
);
```

şi apoi

```
CREATE TABLE studenţi (
matricol ...
...
,spec VARCHAR(50) NOT NULL
REFERENCES specializări (spec)
...
);
```

A doua variantă este mai nimerită, chiar dacă avem un atribut suplimentar de tip cheie surogat prin care identificăm fiecare specializare:

```
CREATE TABLE specializări (
idspec NUMBER(4) NOT NULL PRIMARY KEY
,spec VARCHAR(50)
);

CREATE TABLE studenţi (
matricol ...
...
```

```
,idspec NUMBER(4) NOT NULL
REFERENCES specializări (idspec)
...
);
```

Prețul ultimelor două soluții ține de complexitatea presupusă de gestionarea unei chei străine, precum și de timpul mai mare necesar interogărilor și verificărilor ce decurg din restricția referențială. Marele avantaj ține de flexibilitate, orice specializare nouă sau modificare a denumirii unei specializări neatrăgând modificarea schemei bazei, fiind suficientă doar o comandă DML.

Întrucât folosirea clauzei CHECK prezintă avantajul net al vitezei și simplității (să nu uităm că la restricțiile referențiale apar proleme de genul DELETE RESTRICT/CASCADE și mai ales UPDATE CASCADE/RESTRICT), recomandabil este ca, atunci când lista valorilor acceptate nu e prea lungă, iar acestea (valorile) sunt constante, să recurgem la o regulă de validare la nivel de atribut, în timp ce atunci când valorile sunt mai numeroase și/sau mai susceptibile de a fi modificate, să creăm o nouă relație (și, implicit, o nouă restricție referențială).

Păstrând discuția în baza de date EXAMENE, să vedem ce implicații ar avea o restricție de genul: *fiecare profesor poate preda oricâte discipline, dar numai la specializările care îi sunt repartizate*. Există, prin urmare, profesori ce predau la o singură specializare, profesori ce predau la două specializări etc. Relația DISC_SPEC_PROFI {CodDisc, An, Modul, IdSpec, CodProf} este cea „împricinată", însă aici implementarea acestei restricții printr-o regulă de validare la nivel de înregistrare este cu totul improbabilă, așa că mai realist ar fi să creăm o relație de genul: PROFI_SPECIALIZĂRI {An, Modul, IdSpec, CodProf}. Interesant este că, spre deosebire de exemplul precedent, cheia străină e compusă rău de tot:

```
CREATE TABLE profi_specializări (
 an NUMBER(1) NOT NULL
,modul VARCHAR(20) NOT NULL
,idspec NUMBER(5) NOT NULL REFERENCES specializari(idspec)
,codprof NUMBER(5) NOT NULL REFERENCES profesori (codprof)
,PRIMARY KEY (an, modul, spec, codprof)
);

CREATE TABLE disc_spec_profi (
 coddisc CHAR(6) NOT NULL REFERENCES discipline (coddisc)
,an NUMBER(1) NOT NULL
,modul VARCHAR(20) NOT NULL
,idspec NUMBER(5) NOT NULL REFERENCES specializari(idspec)
,codprof NUMBER(5) NOT NULL REFERENCES profesori (codprof)
,PRIMARY KEY (coddisc, an, modul, spec)
,FOREIGN KEY (an, modul, spec, codprof)
REFERENCES profi_specializări(an, modul, idspec, codprof)
);
```

Oricărei alocări a unei specializări unui profesor îi va corespunde o linie în tabela PROFI_SPECIALIZĂRI. Forma finală al scriptului de creare a tabelelor poate fi descărcată de pe pagina Web cunoscută (*listing*-ul 7.17).

7.3. Dependenţe de incluziune

Este greu de explicat motivul pentru care toate cărţile majore care tratează problema normalizării şi, în general, proiectarea bazelor de date sunt atât de discrete în materie de dependenţe de incluziune. Nu-i vorbă că noţiunea ar fi din cale-afară de dificilă, ci mai degrabă că în aplicaţiile economice, chiar de calibru mediu, este aproape imposibil să nu se manifeste un asemenea gen de dependenţă.

Într-o exprimare lejeră, între două atribute X şi Y există o dependenţă de incluziune (DI) dacă şi numai dacă orice valoare a lui X este obligatoriu şi valoare a lui Y şi se notează simplu $X \subseteq Y$. Mai general, o DI semnifică faptul că proiecţia pe *m* atribute date ale relaţiei R este un subset al proiecţiei pe *m* atribute date din relaţia S. De aici şi caracterizarea DI ca dependenţe interrelaţii[9]. Definiţia însă nu obligă ca R şi S să fie neapărat distincte, deci DI se poate institui între atribute şi grupe de atribute ale aceleiaşi relaţii. După Casanova ş.a., DI semnalizează relaţiile de tip *este un/o*[10], iar Sciore este şi mai limpede când scrie că folosim DI pentru a arăta că două atribute din relaţii diferite se referă la acelaşi lucru[11]. Am putea amenda afirmaţia lui Sciore prin precizarea faptului că atributele nu trebuie neapărat să fie din relaţii diferite.

Cel mai frecvent exemplu de DI este `Manager` $\subseteq$ `Angajat`[12], care înseamnă că orice manager este un angajat al companiei. Graful din figura 7.5 ilustrează o situaţie comună în firmele de pretutindeni: fiecare angajat al unei întreprinderi este arondat unui compartiment (*Producţie*, *Marketing*, *Personal*, *Financiar*, *Contabilitate* etc.); fiecare compartiment are un singur şef; fiecare şef este, şi el, angajat, chiar dacă mai uită uneori de lucrul acesta.

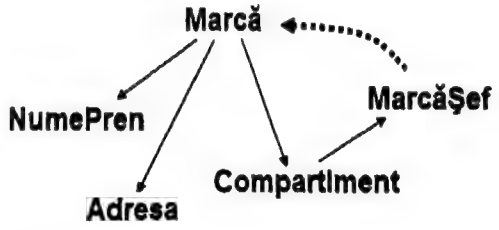


Figura 7.5. O dependenţă de incluziune

Dependenţa de joncţiune reprezentată prin săgeata punctată este deci `MarcăŞef` $\subseteq$ `Marcă`. Decuparea relaţiilor din graf nu ridică mari probleme. Avem două surse de

9. [Fagin 81].
10. [Casanova ş.a. 82].
11. [Sciore 83].
12. [Fagin 81], [Casanova ş.a. 82], [Johnson & Klug 82], [Sciore 83].

DF, prin urmare cele două relaţii vor fi : PERSONAL (Marcă, NumePren, Adresă, Compartiment) şi COMPARTIMENTE (Compartiment, MarcăŞef). Dependenţa de incluziune va fi încorporată în schemă sub forma restricţiei referenţiale dintre COMPARTIMENTE.MarcăŞef şi PERSONAL.Marcă.

Frunze şi conturi

Ne îndepărtăm de exemplele consacrate, apelând la baza de date CONTABILITATE, pe care, după discuţia din paragraful 7.2.2, am adus-o la schema : NOTE_CONTABILE (IdNotăContabilă, Data, ExplicaţiiNotă), OPERAŢIUNI (IdOperaţiune, IdNotăContabilă, ExplicaţiiOp, NrConturiDebitoare, NrConturiCreditoare, UnContDebitor, UnContCreditor) şi DETALII_OPERAŢIUNI (IdOperaţiune, ContDebitor, ContCreditor, Suma). Ei bine, a sosit momentul pentru a introduce una dintre cele mai importante restricţii ale unei aplicaţii de contabilitate generală : conturile care apar în operaţiunile contabile nu trebuie să aibă conturi sintetice/analitice subordonate !

Să începem explicaţiile mai din „amonte". Pentru reflectarea elementelor patrimoniale, a capitalului, datoriilor şi obligaţiilor, contabilitatea foloseşte conturi organizate în serii, clase şi grupe[13]. De exemplu, în seria 1 sunt incluse conturile organice de bilanţ, în seria 2 conturile de procese, în seria 3 conturile de rezultate, în seria 4 conturile în afara bilanţului (extrapatrimoniale), iar în seria 5 conturile privind circuitul contabilităţii manageriale. Seria 1 conţine 7 clase, pentru : capitaluri, active imobilizate, stocuri, terţi, trezorerie, regularizare şi conturi rectificative. Fiecare clasă are una sau mai multe serii de conturi. Astfel, clasa *Stocuri* are şapte grupe : materii şi materiale, obiecte de inventar ; producţie în curs ; produse ; stocuri la terţi ; animale ; mărfuri ; ambalaje. În fiecare grupă sunt conturile propriu-zise, care pot fi sintetice de ordinul 1 (alcătuite din trei cifre) sau de ordinul 2 (patru cifre). Fiecare sintetic de ordinul 2 aparţine unui sintetic de ordinul 1. Tabelul 7.1 indică o porţiune din planul de conturi al firmei X. Orice tip de cont sintetic poate fi descompus pe conturi analitice, în funcţie de specificul şi interesele întreprinderii.

Astfel, contul 301, *Materii prime*, este descompus pe patru analitice. Primele trei, 301.01, 301.02 şi 301.03, reprezintă cele mai importante materii prime pentru firmă, iar ultimul, 301.09, le grupează pe toate celelalte. Mai sunt descompuse pe analitice conturile de *furnizori* (401) şi *cheltuieli cu materiile prime* (600). Interesant că un cont sintetic de gradul I bifuncţional (428) poate avea un „subordonat" de activ (4281) şi un altul de pasiv (4282) !

Din moment ce am devenit mai riguroşi, este evident că, practic, putem introduce în schemă atributele SimbolCont, TipCont, DenumireCont, iar între ContDebitor şi SimbolCont, pe de o parte, şi ContCreditor şi SimbolCont, pe de altă parte, avem de-a face cu o dependenţă de incluziune. Graful dependenţelor ia forma din figura 7.6.

13. Vezi, spre exemplu, Horomnea, E., *Bazele contabilităţii. Concepte, aplicaţii, lexicon*, Editura Sedcom Libris, Iaşi, 2004, pp. 178-180.

Tabelul 7.1. Extras din planul de conturi al unei firme

Simbol cont	TipCont	Denumire cont
Clasa: Conturi de CAPITALURI		
...		
101	pasiv	Capital social
1011	pasiv	Capital social subscris nevărsat
1012	pasiv	Capital social subscris vărsat
...		
211	activ	Terenuri şi amenajări
2111	activ	Terenuri
2112	activ	Amenajări
....		
301	activ	Materii prime
301.01	activ	Nisip
301.02	activ	Ciment
301.03	activ	Var
301.09	activ	Alte materii prime
...		
308	bifuncţional	Diferenţe de preţ la materii prime
...		
401	pasiv	Furnizori
401.01	pasiv	Furnizor A
401.02	pasiv	Furnizor B
401.99	pasiv	Alţi furnizori
...		
428	bifuncţional	Alte creanţe şi datorii faţă de salariaţi
4281	activ	Alte creanţe faţă de salariaţi
4282	pasiv	Alte datorii faţă de salariaţi
...		
411	activ	Clienţi
...		
4426	activ	TVA deductibilă
4427	pasiv	TVA colectată
...		
601	activ	Cheltuieli cu materii prime
601.01	activ	Cheltuieli cu nisipul
601.02	activ	Cheltuieli cu cimentul
601.03	activ	Cheltuieli cu varul
601.09	activ	Cheltuieli cu alte materii prime
...		

Relaţia nouă ce rezultă din graf este evidentă : PLAN_CONTURI (`SimbolCont`, `TipCont`, `DenumireCont`), iar cele două dependenţe de incluziune se vor materializa la implementarea bazei în două restricţii referenţiale (vezi *listing*-ul 7.18 de pe pagina Web).

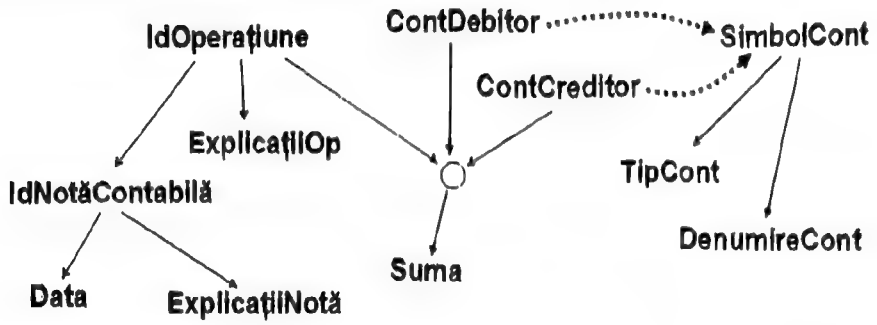


Figura 7.6. Dependenţele funcţionale şi de incluziune pentru BD CONTABILITATE

Revenind la restricţia pe care am definit-o şi caracterizat-o drept fundamentală într-o aplicaţie destinată contabilităţii generale, dacă avem în vedere structura arborescentă a planului de conturi, putem reformula cerinţa astfel : orice cont debitor şi creditor ce apare într-o înregistrare contabilă trebuie să fie „frunză", adică nu poate avea conturi (sintetice de ordinul 2 sau analitice) subordonate. Aşadar, declanşatoarele de inserare şi modificare ale tabelei DETALII_OPERAŢIUNI trebuie să verifice calitatea de frunză pentru `ContDebitor` şi `ContCreditor`.

Continuăm cu declanşatorul de inserare pe care l-am conturat iniţial în paragraful 7.2.2 – vezi *listing*-ul 7.19.

Listing 7.19. Un declanşator de inserare nou-nouţ pentru tabela DETALII_OPERAŢIUNI

```
CREATE OR REPLACE TRIGGER trg_do_ins
AFTER INSERT ON detalii_operatiuni FOR EACH ROW
DECLARE
v_nr NUMBER(3) := 0 ;
BEGIN
-- se numara câte conturi încep cu simbolurile contului
-- debitor, pentru a vedea daca acesta se descompune
SELECT COUNT(*) INTO v_nr FROM plan_conturi
WHERE SUBSTR(SimbolCont,1,LENGTH(:NEW.ContDebitor))
  = :NEW.ContDebitor ;
IF v_nr > 1 THEN
   RAISE_APPLICATION_ERROR(-20194,
        'EROARE! ContDebitor nu este FRUNZA!!!') ;
END IF ;

-- se numara câte conturi încep cu simbolurile contului
-- creditor, pentru a vedea daca acesta se descompune
SELECT COUNT(*) INTO v_nr FROM plan_conturi
WHERE SUBSTR(SimbolCont,1,LENGTH(:NEW.ContCreditor))
  = :NEW.ContCreditor ;
IF v_nr > 1 THEN
    RAISE_APPLICATION_ERROR(-20195,
        'EROARE! ContCreditor nu este FRUNZA!!!') ;
END IF ;
```

```
UPDATE operatiuni
SET  nrconturidebitoare =  NVL(nrconturidebitoare, 1) +
        CASE WHEN :NEW.contdebitor <>
                NVL(uncontdebitor,:NEW.contdebitor)
     THEN 1 ELSE 0 END,
     nrconturicreditoare =  NVL(nrconturicreditoare, 1) +
       CASE WHEN :NEW.contcreditor <>
              NVL(uncontcreditor,:NEW.contcreditor)
      THEN 1 ELSE 0 END,
     uncontdebitor = :NEW.contdebitor,
     uncontcreditor = :NEW.contcreditor
  WHERE idoperatiune = :NEW.idoperatiune ;
END ;
```

Rezolvarea poate fi ameliorată, dacă de gândim că fiecare inserare a unei linii în DETALII_OPERAŢIUNI atrage după sine două rânduri de scanare (pentru numărare) a întregului plan de conturi. Avem deja o oarecare experienţă în lucrul cu atribute redundate, aşa încât cel mai la îndemână ar fi să adăugăm în PLAN_CONTURI un câmp de tip BOOLEAN (logic), pe care să-l numim chiar `EsteFrunză`, astfel încât noua structură a tabelei va fi PLAN_CONTURI {`SimbolCont`, `TipCont`, `DenumireCont`, `EsteFrunză`}. Soluţia Oracle este însă:

```
ALTER TABLE plan_conturi ADD (EsteFrunza CHAR(1)
   DEFAULT 'D' CHECK (EsteFrunza IN ('D', 'N'))) ;
```

Deoarece, din păcate, în Oracle un atribut nu poate avea tipul BOOLEAN, am recurs la tipul CHAR(1), limitând valorile la „D" (da) şi „N" (nu).

Declanşatoarele tabelei vor „veghea" la corectitudinea acestui atribut. Astfel, când se inserează o linie în PLAN_CONTURI se verifică dacă noul cont *nu* este sintetic de ordinul 2 sau analitic. Dacă da (de exemplu: 301.01), se verifică dacă superiorul său (301, în exemplul nostru) era „frunză" înainte de inserare. În caz că există măcar o înregistrare contabilă în care „superiorul" apare pe debit sau pe credit, inserarea în planul de conturi este interzisă, întrucât este imposibil de revenit asupra notelor contabile deja existente şi a le „re-conta" (vezi *listing*-ul 7.20).

Listing 7.20. Declanşatorul de inserare în PLAN_CONTURI

```
CREATE OR REPLACE TRIGGER trg_pc_ins
 BEFORE INSERT ON plan_conturi FOR EACH ROW
DECLARE
v_parinte VARCHAR(15) ;
v_gata BOOLEAN := FALSE ;
v_nr NUMBER(3) := 0 ;
BEGIN
IF LENGTH(:NEW.SimbolCont) > 3 THEN
  --contul este sintetic de ordin 2 sau analitic

  -- se încearca gasirea parintelui, taind câte un caracter
```

```
    v_parinte := :NEW.SimbolCont ;
    WHILE v_gata=FALSE AND LENGTH(v_parinte)>3 LOOP
        v_parinte := SUBSTR(v_parinte,1, LENGTH(v_parinte) - 1);
    SELECT COUNT(*) INTO v_nr FROM plan_conturi
        WHERE SimbolCont=v_parinte ;
    IF NVL(v_nr,0) = 1 THEN
            v_gata := TRUE ;
        EXIT ;
    END IF ;
    END LOOP ;
    IF v_gata THEN
        -- exista un parinte pentru noul cont
    -- testam daca parintele este frunza
        SELECT COUNT(ContDebitor)+ COUNT(ContCreditor)
        INTO v_nr FROM detalii_operatiuni
        WHERE ContDebitor=v_parinte OR ContCreditor=v_parinte ;
    IF NVL(v_nr,0) >= 1 THEN
        RAISE_APPLICATION_ERROR (-20196,
            'EROARE! Contul parinte apare deja in operatiuni,'||
         ' iar noul cont nu mai poate fi adaugat!!!') ;
        END IF ;
    UPDATE plan_conturi SET EsteFrunza = 'N'
        WHERE SimbolCont = v_parinte ;
    END IF;
END IF ;
:NEW.EsteFrunza := 'D' ;
END ;
```

Acum ştim că, atunci când adăugăm un cont, se actualizează automat atributul `EsteFrunză` pentru contul părinte. Pentru asigurarea corectitudinii valorilor acestui atribut, trebuie realizate şi declanşatoarele pentru modificare şi ştergere.

Trecem acum la declanşatoarele tabelei DETALII_OPERAŢIUNI. Mai întâi, creăm o funcţie (*listing*-ul 7.21) care primeşte drept parametru simbolul unui cont şi returnează valoarea atributului `EsteFrunză` pentru contul respectiv.

Listing 7.21. Funcţia PL/SQL F_ESTE_FRUNZA

```
CREATE OR REPLACE FUNCTION f_este_frunza (simbol_cont VARCHAR2)
    RETURN BOOLEAN
AS
    v_frunza CHAR(1) ;
BEGIN
SELECT EsteFrunza INTO v_frunza FROM plan_conturi
 WHERE SimbolCont = simbol_cont ;
IF v_frunza = 'D' THEN
    RETURN TRUE ;
ELSE
    RETURN FALSE ;
END IF ;
END ;
```

Sarcina declanşatoarelor se simplifică sensibil. Iată-l pe cel de inserare (*listing*-ul 7.22).

Listing 7.22. Noul declanşator de inserare în DETALII_OPERAŢIUNI

```
CREATE OR REPLACE TRIGGER trg_do_ins
 AFTER INSERT ON detalii_operatiuni FOR EACH ROW
DECLARE
v_nr NUMBER(3) := 0 ;
BEGIN
IF f_este_frunza (:NEW.ContDebitor)=FALSE OR
           f_este_frunza (:NEW.ContCreditor)=FALSE THEN
   -- EROARE! Cel putin unul dintre conturile
   -- de pe debit sau credit nu este FRUNZA!!!
  RAISE_APPLICATION_ERROR(-20194,
  'EROARE! Contul de pe debit sau credit nu este FRUNZA!!!') ;
END IF ;
UPDATE operatiuni
SET  nrconturidebitoare =  NVL(nrconturidebitoare, 1) +
    CASE WHEN :NEW.contdebitor <>
           NVL(uncontdebitor,:NEW.contdebitor)
     THEN 1  ELSE 0 END,
     nrconturicreditoare =  NVL(nrconturicreditoare, 1) + CASE
     WHEN :NEW.contcreditor <>
           NVL(uncontcreditor,:NEW.contcreditor)
     THEN 1 ELSE 0 END,
    uncontdebitor = :NEW.contdebitor,
    uncontcreditor = :NEW.contcreditor
  WHERE idoperatiune = :NEW.idoperatiune ;
END ;
```

Proiecte vechi şi noi

Am recurs la un exemplu privind proiectele derulate într-o firmă cu ocazia paragrafului 5.4, pentru a ilustra forma normalizată Boyce-Codd. Acum însă, ne apropiem de realitatea dintr-o companie, luând în calcul următoarele elemente:

- un angajat, identificat prin `Marcă`, este inclus în schema unui compartiment funcţional (*Contabilitate*, *Marketing* etc.);
- orice compartiment are un singur şef (`MarcăŞef`);
- firma, datorită specificului activităţii sale, lucrează pe bază de proiecte; fiecare proiect are un identificator (`IdProiect`), demarează la o anumită dată (`DatăProiect`) şi are un termen de realizare (`Termen`);
- un proiect este alcătuit din una sau mai multe activităţi (`IdActivitate`), o activitate având o titulatură (`DenActivit`) şi o scurtă descriere (`DescActivit`);
- orice angajat este capabil să desfăşoare una sau mai multe activităţi;
- un angajat poate lucra la mai multe proiecte;
- la un proiect sunt angajate una sau mai multe persoane;
- o activitate poate fi inclusă în unul sau mai multe proiecte;
- în cadrul unui proiect, fiecare activitate începe la o anumită dată (`DataÎnceput`), se finalizează la o altă dată (`DataFinal`) şi constă într-o serie de operaţii, responsabilităţi şi rezultate (`Detalii`);

- o aceeaşi persoană poate desfăşura, în acelaşi proiect, una, două sau mai multe activităţi;
- o activitate într-un proiect poate fi desfăşurată de mai multe persoane.

Dependenţele corespunzătoare acestor cerinţe pot fi reprezentate sub formă de graf, ca în figura 7.7.

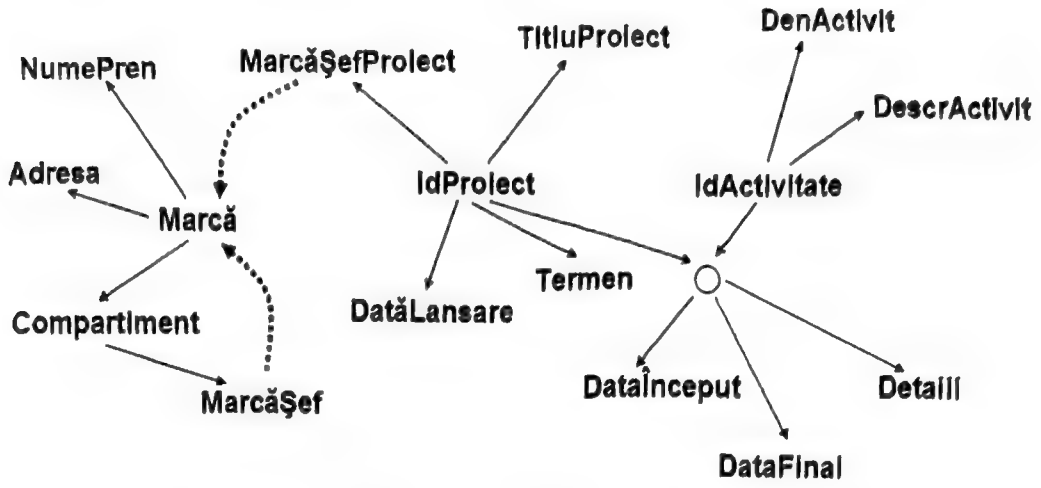


Figura 7.7. Dependenţele bazei de date PROIECTE

Paragraful 5.5.3 ne-a introdus într-una dintre cele mai delicate probleme ale normalizării, şi anume imposibilitatea de a prelua sub formă de dependenţe anumite relaţii semantice. Şi graful de mai sus are câteva asemenea dureri de cap. Mai precis, există o serie de informaţii care nu pot fi furnizate de schema construită pe baza decupării relaţiilor:

- care sunt activităţile pe care le poate desfăşura o persoană?
- în ce proiecte a fost implicat un angajat?
- care sunt persoanele implicate într-un anumit proiect?
- care este activitatea sau care sunt activităţile desfăşurate de o anumită persoană într-un proiect dat?
- care sunt persoanele care au desfăşurat o anumită activitate într-un anumit proiect?

Sugestia din paragraful 5.5.3 era de a introduce în schema bazei şi alte atribute care să „fixeze" toate legăturile neprinse în dependenţe. Astfel, un angajat poate fi „legat" de o anumită activitate pe care o poate desfăşura printr-un atribut nou, `Competenţe`, care să sugereze abilităţile şi experienţa angajatului în privinţa activităţii cu pricina: (`Marcă, IdActivitate`) ⟶ `Competenţe`. Pe de altă parte, participarea unui angajat într-o activitate din cadrul unui proiect poate fi descrisă cu ajutorul a trei atribute – `Atribuţii, Probleme` şi `Rezultate` –, care indică ce sarcini a avut angajatul legat de acea activitate din cadrul proiectului, ce probleme a întâmpinat şi care au fost rezultatele muncii sale. Dependenţele sunt evidente: (`Marcă, IdProiect, IdActivitate`) ⟶ `Atribuţii`, (`Marcă, IdProiect, IdActivitate`) ⟶ `Probleme` şi (`Marcă, IdProiect, IdActivitate`) ⟶

`Rezultate`. Noul graf s-ar prezenta ca în figura 7.8, noile atribute fiind scrise „aplecat" (italic).

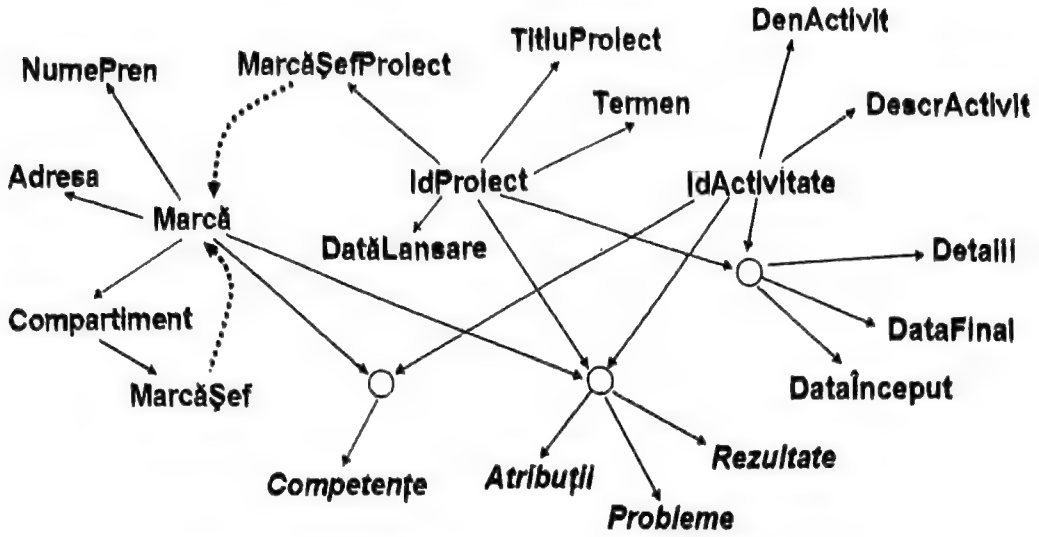


Figura 7.8. Noul graf al dependenţelor BD PROIECTE

Relaţiile decupate din graf sunt următoarele:

PERSONAL {Marcă, Numepren, Adresă, Compartiment}
COMPARTIMENTE {Compartiment, MarcăŞef}
ACTIVITĂŢI {IdActivitate, DenActivit, DescrActivit}
COMPETENŢE {Marcă, IdActivitate, Competenţe}
PROIECTE {IdProiect, TitluProiect, MarcăŞefProiect, DataLansare, Termen}
ACTIVITĂŢI_PROIECTE {IdProiect, IdActivitate, DataÎnceput, DataFinal, Detalii}
PERSOANE_ACTIVITĂŢI_PROIECTE {IdProiect, IdActivitate, Marcă, Atribuţii, Probleme, Rezultate}

Alte frunze, dar aceleaşi conturi

Soluţiei la care am ajuns în acest paragraf pentru schema bazei de date CONTABILITATE i se poate reproşa, printre altele, faptul că, uneori, la inserarea unei linii în PLAN_CONTURI trebuie modificate alte linii din aceeaşi tabelă, ceea ce nu este posibil în toate SGBD-urile. Iar dacă la inserare lucrurile mai pot funcţiona, declanşatorul de modificare are şanse considerabile să se împotmolească.

Pentru a încerca să ajungem la următoarea soluţie, pornim de la dependenţele conţinute în graful din figura 7.6. Dependenţele de incluziune dintre `ContDebitor`, pe de o parte, `ContCreditor`, pe de altă parte, şi `SimbolCont` sunt discutabile. De fapt, restricţia este ca toate conturile debitoare şi creditoare din orice înregistrare contabilă să fie elementare (frunze), adică să nu fie descompuse pe analitice (sau sintetice de ordinul 2). Putem defini deci un atribut denumit `ContElementar` care identifică (printr-o valoare booleană *TRUE* sau un şir de caractere *Da*) un cont

„frunză". Putem vorbi de o „specializare" a conturilor, cele două dependenţe de incluziune iniţiale fiind acum `ContDebitor` ⊆ `ContElementar`, respectiv `ContCreditor` ⊆ `ContElementar`, la care se adaugă `ContElementar` ⊆ `SimbolCont`. Spre deosebire de celelalte două, ultima DI indică o specializare. Noul graf al dependenţelor este cel din figura 7.9.

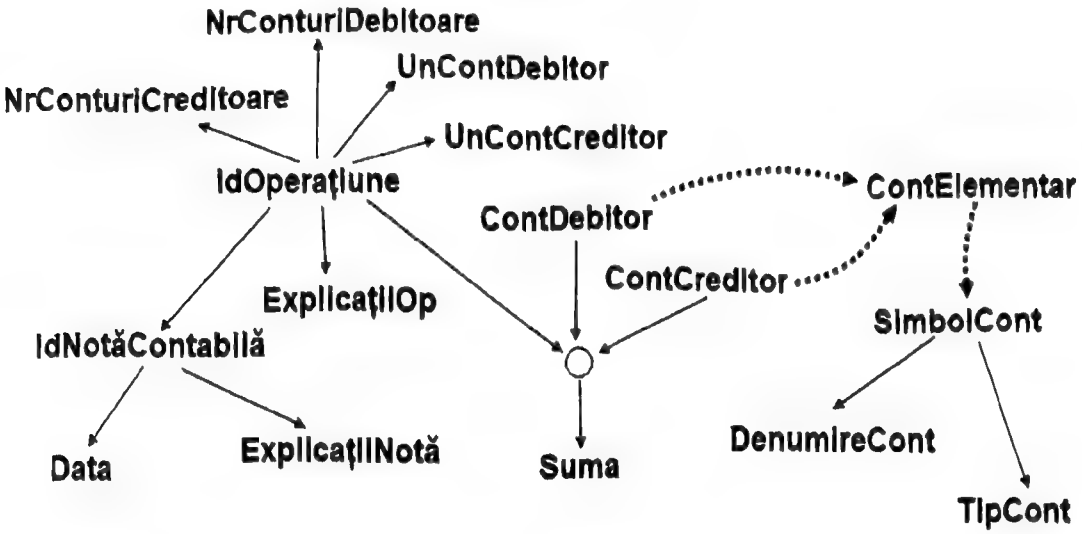


Figura 7.9. Trei dependenţe de incluziune, dintre care una semnifică o specializare

Decupând relaţiile din graf, obţinem:

PLAN_CONTURI {SimbolCont, TipCont, DenumireCont}
CONTURI_ELEMENTARE {ContElementar}
NOTE_CONTABILE {IdNotăContabilă, Data, ExplicaţiiNotă}
OPERAŢIUNI {IdOperaţiune, IdNotăContabilă, ExplicaţiiOp, NrConturiDebitoare, NrConturiCreditoare, UnContDebitor, UnContCreditor}
DETALII_OPERAŢIUNI {IdOperaţiune, ContDebitor, ContCreditor, Suma}

Lucrurile sunt chiar interesante, deoarece acum restricţiile referenţiale se stabilesc astfel:

- între `DETALII_OPERAŢIUNI.ContDebitor` (copil) şi `CONTURI_ELEMENTARE.ContElementar` (părinte);
- între `DETALII_OPERAŢIUNI.ContCreditor` (copil) şi `CONTURI_ELEMENTARE.ContElementar` (părinte);
- între `CONTURI_ELEMENTARE.ContElementar` (copil) şi `PLAN_CONTURI.SimbolCont`.

Prin urmare, specializarea se poate traduce, în acest caz, printr-o nouă relaţie cu un singur atribut. *Listing*-ul 7.23 pentru recrearea bazei de date CONTABILITATE poate fi descărcat de la adresa Web menţionată cu alte prilejuri.

Şi declanşatoarele tabelei PLAN_CONTURI se simplifică în mare măsură. Pentru comparaţie, îl vom discuta doar pe cel de inserare. Dar, mai înainte, să creăm o

funcţie (*listing*-ul 7.24) care să primească drept parametru un cont şi care să returneze TRUE dacă acest cont apare măcar într-o operaţiune contabilă (pe debit sau credit) şi FALSE în caz contrar:

Listing 7.24. Funcţia F_APARE

```
CREATE OR REPLACE FUNCTION f_apare (
   simbol_cont plan_conturi.SimbolCont%TYPE)
RETURN BOOLEAN
AS
v_unu NUMBER(1) ;
BEGIN
SELECT 1 INTO v_unu FROM dual WHERE EXISTS (
  SELECT 1 FROM detalii_operatiuni
  WHERE ContDebitor=simbol_cont OR ContCreditor=simbol_cont);
RETURN TRUE ;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN FALSE ;
END ;
```

Funcţia ne este de folos în declanşatorul de inserare în PLAN_CONTURI (*listing*-ul 7.25) pentru a verifica dacă părintele contului inserat a fost debitat sau creditat în operaţiuni contabile:

Listing 7.25. Noul declanşator de inserare al PLAN_CONTURI

```
CREATE OR REPLACE TRIGGER trg_pc_ins
 AFTER INSERT ON plan_conturi FOR EACH ROW
DECLARE
v_parinte VARCHAR(15) ;
v_gata BOOLEAN := FALSE ;
v_nr NUMBER(3) := 0 ;
BEGIN
v_parinte := SUBSTR(:NEW.SimbolCont,1, LENGTH(:NEW.SimbolCont)-1);

WHILE LENGTH(v_parinte) > 3 LOOP
   DELETE FROM conturi_elementare
   WHERE ContElementar = v_parinte ;

   IF SQL%ROWCOUNT > 0 THEN -- exista un parinte-frunza
      IF f_apare (v_parinte) THEN
      RAISE_APPLICATION_ERROR(-20193,
         'EROARE! Nu se poate adauga contul dorit!!!') ;
      END IF ;
   END IF;
   v_parinte := SUBSTR(v_parinte,1,LENGTH(v_parinte)-1);
END LOOP ;
-- noul cont este frunza!!!
INSERT INTO conturi_elementare VALUES (:NEW.SimbolCont) ;
END ;
```

Cu noua structură se schimbă şi funcţia f_este_frunză: vezi *listing*-ul 7.26.

Listing 7.26. Noul conţinut al funcţiei f_este_frunză

```
CREATE OR REPLACE FUNCTION f_este_frunza (simbol_cont VARCHAR2)
RETURN BOOLEAN
AS
v_unu NUMBER(1) ;
BEGIN
SELECT 1 INTO v_unu FROM DUAL WHERE EXISTS
 (SELECT 1 FROM conturi_elementare
    WHERE ContElementar = simbol_cont) ;
RETURN TRUE ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RETURN FALSE ;
END ;
```

În schimb, declanşatorul de inserare în DETALII_OPERAŢIUNI poate rămâne neschimbat.

7.4. Caz practic – centru de închiriere a casetelor video

Deşi nu întotdeauna la zi cu legislaţia pentru protecţia drepturilor de autor, centrele de închiriere de casete video au constituit o afacere tentantă, mai ales în anii '90 şi, am putea spune, îşi are încă clientela sa, deşi HBO, în general televiziunea prin cablu, constituie un concurent foarte puternic. Pentru a-l putea diferenţia de concurenţă, vrem ca baza de date a centrului pentru care lucrăm să ofere şi date care să determine fidelitatea clienţilor:

- informaţii despre filme: genul de film, distribuţie, realizare (scenariu, regie) etc.;
- premiile importante luate de filme: premiul, categoria, anul decernării;
- date despre împrumuturi, restituiri şi întârzieri;
- informaţii statistice, de genul: filmele cele mai cerute, clienţii cei mai fideli, clienţii cu cele mai mari întârzieri la returnarea casetelor;
- date despre clienţi: zi de naştere/onomastică, genuri preferate şi, implicit, structura pe vârste şi ocupaţii ale clienţilor;
- un sistem de evaluare (*rating*, în termeni elevaţi) a filmelor de către clienţi.

Ceea ce am discutat despre baza de date FILMOGRAFIE în capitolele anterioare ne este deci de mare folos pentru cele ce urmează. În privinţa atributelor surogat, de la început este evident că am avea nevoie de cel puţin trei:

- `IdFilm` – pentru identificarea fără ambiguitate a unui film;
- `IdCasetă` – un număr unic pentru diferenţierea între ele a casetelor;
- `IdÎmprumut` – un număr atribuit fiecărui împrumut de una sau mai multe casete, la un moment oarecare.

Pentru identificarea clienţilor am putea să considerăm codul numeric personal (CNP-ul) ca fiind mulţumitor.

Din paragraful 6.2 ne-au rămas câteva dependenţe memorabile, unele funcţionale:

(1) `IdFilm` ⟶ `TitluOriginal`

(2) `IdFilm` ⟶ `TitluRO`

(3) `IdFilm` ⟶ `AnLans`

(4) `DenPremiu` ⟶ `LocDecernare`

(5) (`IdFilm`, `Rol`) ⟶ `Actor`

(6) (`DenPremiu`, `Categorie`, `IdFilm`) ⟶ `AnPremiu`

(7) (`DenPremiu`, `Categorie`, `IdFilm`, `Actor`) ⟶ `AnPremiu`

iar altele multivalorice:

`IdFilm` ⟶→ `Producător` | `Regizor`

`IdFilm` ⟶→ `Producător` | `Gen` sau `IdFilm` ⟶→ `Regizor` | `Gen`

Adăugăm şi scenaristul, aşa că:

`IdFilm` ⟶→ `Producător` | `Scenarist` sau `IdFilm` ⟶→ `Regizor` | `Gen`

Caseta este identificată de atributul `IdCasetă`, astfel că putem scrie:

(8) `IdCasetă` ⟶ `DataCumpărării`

(9) `IdCasetă` ⟶ `ProducătorCasetă`

(10) `IdCasetă` ⟶ `AnProdCasetă`

(11) `IdCasetă` ⟶ `PreţCumpărare`

Anticipând o cerere mai mare, un film poate fi cumpărat în mai multe exemplare, deci pe mai multe casete: `IdFilm` —/→ `IdCasetă`. Nici reciproca nu e prea valabilă, întrucât pe o casetă pot fi adunate filme de mai mică dimensiune, scurtmetraje (doar trei exemple: filmuleţele cu Stan şi Bran, Chaplin şi Tom şi Jerry): `IdCasetă` —/→ `IdFilm`.

Pentru a putea gestiona coerent toate fimele (scurtmetraje sau nu) de pe o casetă, apelăm la un atribut de genul `FilmNr`, adică numărul de ordine al filmului de pe o casetă (vă amintiţi de atributul `Linie` pentru facturi?). Aşa că:

(12) (`IdCasetă`, `FilmNr`) ⟶ `IdFilm`

Am zice că problema raportului filme-casete este tranşată. Da' de unde! Persoanele mai simţitoare poate-şi amintesc de febra pre-telenovelistă declanşată de un film epopee - *Pasărea Spin.* Ei bine, ca şi alte filme-maraton, acesta se întinde pe mai multe casete. Astfel încât, spre exemplu, pe o casetă poate fi partea a patra a filmului XYZ. Din fericire, dependenţa de mai sus nu este compromisă, însă am avea nevoie de un atribut adiţional - `ParteFilm`, iar dependenţa s-ar putea scrie:

(13) (`IdCasetă`, `FilmNr`) ⟶ `ParteFilm`

Ba chiar putem fi siguri că, pe o casetă, casa producătoare poate să introducă ultima parte a unui film plus un scurt-metraj de acelaşi regizor sau un medalion actoricesc etc.

Despre clienţi, numai de bine:

(14) `CNPClient` ⟶ `NumeClient`

(15) `CNPClient` ⟶ `AdresaClient`

(16) `CNPClient` ⟶ `TelefonClient`

(17) `CNPClient` ⟶ `DataNaştereClient`

(18) `CNPClient` ⟶ `NivelStudiiClient`

Un client poate împrumuta una sau mai multe casete. Fiecare împrumut este identificat printr-o cheie surogat - `IdÎmprumut`, aşa că:

(19) `IdÎmprumut` ⟶ `DataOraImprumut`

(20) `IdÎmprumut` ⟶ `CNPClient`

Deoarece se pot împrumuta mai multe casete simultan, `IdÎmprumut` —/→ `IdCasetă`, ceea ce e destul de neliniştitor, atâta vreme cât obiectivul central al aplicaţiei este de a gestiona împrumuturile de casete. Firește, prima tentaţie este de a recurge la un atribut suplimentar, ca în cazul liniilor din facturi, atribut căruia îi putem spune chiar `NrCrtÎmpr` (adică un soi de număr curent al împrumutului, număr care indică a câta casetă din împrumut este cea curentă), aşa că, cu un pic de cârpeală, problema s-ar rezolva:

(`IdÎmprumut`, `NrCrtÎmpr`) ⟶ `IdCasetă`

Dacă adăugăm că restituirea casetelor nu este întotdeauna simultană, adică un client poate împrumuta trei casete într-o sâmbătă dimineaţă şi să returneze una sâmbătă seara şi pe celelalte două duminică, se poate scrie, fie (`IdÎmprumut`, `NrCrtÎmpr`) ⟶ `DataOraRestituirii`, fie (`IdÎmprumut`, `IdCasetă`) ⟶ `DataOraRestituirii`. Eu, unul, aş alege varianta din urmă.

Ei bine, dacă privim mai atent ultima dependenţă, observăm că, la o adică, aceasta ar face de prisos atributul `NrCrtÎmpr`. Diferenţa dintre varianta folosirii atributului `NrCrtÎmpr` (varianta 1) şi cea fără (a doua) este ilustrată în figura 7.10.

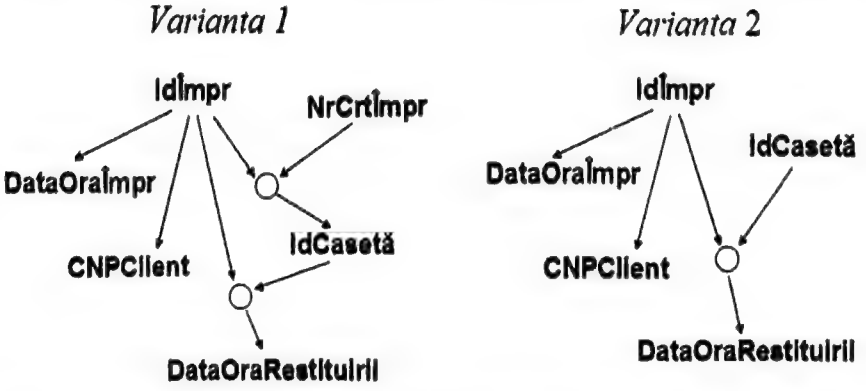


Figura 7.10. Două variante de dependenţe pentru împrumuturi şi restituiri

Fiecare îşi are avantajele şi dezavantajele sale. Prima pare mai complicată şi, totodată, are un uşor aer de artificialitate indus de folosirea `NrCrtÎmpr`. Cel mai important avantaj al său ţine de faptul că preia foarte bine succesiunea temporală împrumut (prima DF cu sursa compusă) – restituire (a doua). Consecinţa esenţială este că în schemă nu apar valori nule. Graful din dreapta figurii este mult mai simplu. Din momentul împrumutului până în cel al restituirii, valoarea atributului `DataOraRestituirii` ar fi NULL. Astfel, pentru a afla în orice moment casetele aflate la clienţi, condiţia în SQL ar fi DataOraRestuirii IS NULL. Tocmai adversarii folosirii valorilor nule ar dezaproba această a doua soluţie. Noi însă, nefiind duşmani (dar nici prieteni) ai nulităţii în bazele de date relaţionale, vom opta pentru această a doua soluţie, chiar dacă lenea a cântărit mult în alegere. Aşadar:

(21) (`IdÎmprumut`, `IdCasetă`) ⟶ `DataOraRestituirii`

Să adunăm tot ce-am discutat până acum într-un graf. Figura 7.11. este o reprezentare destul de bună a ansamblului de dependenţe funcţionale (1)-(21), plus cele multivaloare. Este drept, aglomeraţia este cam mare, dar arta cere sacrificii.

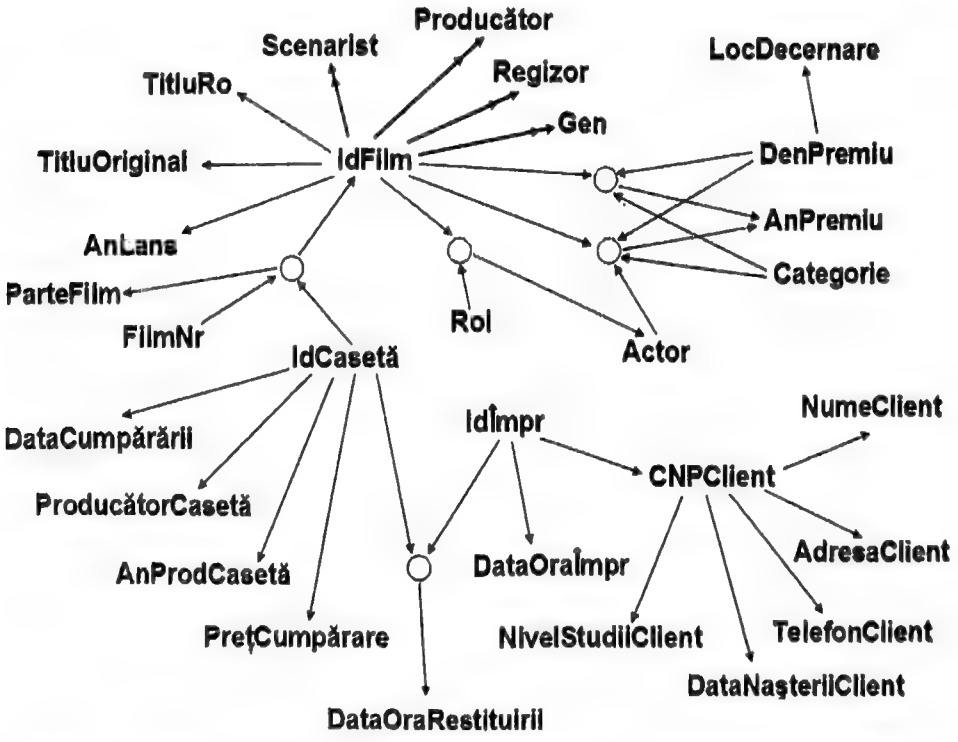


Figura 7.11. Graful DF pentru BD a centrului de închiriere – versiunea 1.0

Decupăm relaţiile din graf:

PREMII_DENUMIRI {`DenPremiu`, `LocDecernare`}
FILME {`IdFilm`, `TitluOriginal`, `TitluRO`, `AnLans`}
DISTRIBUŢIE {`IdFilm`, `Rol`, `Actor`}
PREMII_FILME {`IdFilm`, `DenPremiu`, `Categorie`, `AnPremiu`}
PREMII_INTERPRETARE {`IdFilm`, `Actor` , `DenPremiu`, `Categorie`, `AnPremiu`}
PRODUCĂTORI {`IdFilm`, `Producător`}

REGIZORI (IdFilm, Regizor)
SCENARIȘTI (IdFilm, Scenarist)
FILME_GENURI (IdFilm, Gen)
CASETE (IdCasetă, DataCumpărării, ProducătorCasetă, AnProdCasetă, PreţCumpărare)
CASETE_FILME (IdCasetă, FilmNr, IdFilm, ParteFilm)
CLIENȚI (CNPClient, NumeClient, AdresaClient, TelefonClient, DataNașteriiClient, NivelStudiiClient)
ÎMPRUMUTURI (IdÎmpr, DataOraÎmpr, CNPClient)
CASETE_ÎMPRUMUTATE (IdÎmpr, IdCasetă, DataOraRestituirii)

Schema pare a răspunde necesarului informaţional pe care ni-l propusesem iniţial. Astfel, putem afla: numărul de casete împrumutate de un client pe o anumită perioadă; ponderea comediilor dintre filmele închiriate de un client sau pe tot centrul; ce genuri de filme preferă adulţii între 30 şi 35 de ani cu studii medii etc.

Să ne ocupăm însă de regulile scrise sau nescrise ale centrului:

- un client nu poate împrumuta mai mult de patru casete simultan!
- la fiecare 10 casete împrumutate, un client are dreptul la o casetă împrumutată gratuit;
- împrumutul este pentru 48 de ore; la remiterea casetei se calculează o penalizare de 75% din preţul de închiriere al casetei pentru fiecare zi de întârziere; iată şi tarifele zilnice la închiriere:
  - 50 000 lei pentru casetele produse în anul calendaristic curent şi în cel precedent;
  - 40 000 lei pentru casetele produse în urmă cu doi şi trei ani;
  - 30 000 lei pentru restul;
- pierderea sau distrugerea unei casete antrenează o amendă care reprezintă dublul preţului casetei.

Prima regulă s-ar implementa în schema actuală a bazei în maniera următoare: atunci când în tabela CASETE_ÎMPRUMUTATE se introduce o linie nouă, pentru care valorile ar fi notate cu (*:NEW.IdImpr*, *:NEW.IdCaseta*, *NULL*), după un calapod similar Oracle, se numără, pentru clientul aferent împrumutului curent, câte dintre casetele împrumutate au valoarea atributului DataOraRestituirii nulă:

```
SELECT COUNT(*)
INTO v_CaseteNerestituite
FROM casete_imprumutate c INNER JOIN imprumuturi i
   ON c.idimpr=i.idimpr
WHERE DataOraRestituirii IS NULL AND CNPClient IN
   (SELECT CNPClient
    FROM  imprumuturi
    WHERE idimpr = :NEW.idimpr)
```

Ulterior, valoarea variabilei se testează dacă este mai mică decât patru, caz în care inserarea este permisă; altminteri, inserarea se respinge. Fraza SELECT de mai

sus poate fi folosită în formularul de lucru (care rulează pe platforma client), într-o secvenţă de cod de pe serverul de aplicaţii (dacă arhitectura este de tip Web, adică pe trei sau mai multe straturi), dar cel mai sigur ar fi să se recurgă la un declanşator pe tip `INSERT` pentru această tabelă. Bineînţeles, numai dacă SGBD-ul folosit „suportă" declanşatoare (nu este cazul MySQL-ului, din păcate, decât în câteva versiuni comerciale).

Ei bine, recomandarea noastră este nu numai să se folosească un SGBD ce permite lucrul cu declanşatoare, dar să se introducă un atribut redundant numit `NrCaseteNerestituite`, atribut actualizabil la declanşatoarele de inserare, modificare şi ştergere în CASETE_ÎMPRUMUTATE. Acest atribut ar depinde de `CNPClient`: `CNPClient` ⟶ `NrCaseteNerestituite`.

Ba chiar am putea exagera şi mai mult, folosind două atribute dependente de `CNPClient`, unul `NrCaseteÎmprumutate` şi altul `NrCaseteRestituite`. Paradoxal, deşi ideea pare deplasată, există argumente în favoarea sa. Iar, întrucât a doua regulă spune că *la fiecare 10 casete împrumutate, un client are dreptul la o casetă împrumutată gratuit*, chiar ne convin ambele atribute, aşa că, în dezaprobarea publicului, ne precipităm şi scriem:

(22) `CNPClient` ⟶ `NrCaseteÎmprumutate`

(23) `CNPClient` ⟶ `NrCaseteRestituite`

Mecanismul de actualizare este unul comun pentru utilizatorii de declanşatoare (*triggers*): `NrCaseteÎmprumutate` se incrementează la inserarea unei linii în CASETE_ÎMPRUMUTATE, iar `NrCaseteRestituite` la modificarea unei linii în CASETE_ÎMPRUMUTATE, atunci când condiţia îndeplinită este `:OLD.DataOraRestituirii IS NULL AND :NEW.DataOraRestituirii IS NOT NULL`. Explicaţia condiţiei este una simplă: restituirea unei casete împrumutate se consemnează în baza noastră de date prin, să-i zicem, „de-nulizarea" atributului `CASETE_ÎMPRUMUTATE.DataOraRestituirii`.

A doua regulă a centrului reprezintă o tentativă de a determina fidelitatea clienţilor, mai ales a celor dependenţi: *la 10 casete închiriate, a 11-a este împrumutată gratuit*. Noroc de atributul `NrCaseteÎmprumutate` de mai sus, pentru că, în condiţiile în care acesta este actualizat corect, putem şti în orice moment dacă se cuvine să acordăm gratuitatea sau nu. Pentru a cunoaşte totuşi, în timp, câte casete au fost împrumutate cu titlu gratuit, am putea recurge la un atribut nou, `Gratuită`, care să ia valorile „D" (da) sau „N" şi să depindă funcţional astfel:

(24) (`IdÎmprumut`, `IdCasetă`) ⟶ `Gratuită`

A treia regulă e ceva mai dură: *împrumutul este pentru 48 de ore; la remiterea casetei se calculează o penalizare de 75% din preţul casetei pentru fiecare zi de întârziere*. Implementarea sa ar presupune ca, la „de-nulizarea" valorii unui atribut `DataOraRestituirii`, să verificăm dacă diferenţa `CASETE_ÎMPRUMUTATE.DataOraRestituirii-ÎMPRUMUTURI.DataÎmpr` este mai mare de două zile (întrucât cele două atribute sunt de tip DATETIME, adică dată şi oră, diferenţa lor furnizează, de obicei, numărul de zile dintre cele două momente). Dacă da, atunci calculăm

penalizarea după relaţia: `Penalizare := 0.75 * (CASETE_ÎMPRUMUTATE.DataOraRestituirii-ÎMPRUMUTURI.DataÎmpr-2)`. Este evidentă nevoia de atributul `Penalizare`, care va depinde funcţional astfel:

(25) (`IdÎmprumut`, `IdCasetă`) ⟶ `Penalizare`

Cu această ocazie, ne putem propune să rezolvăm o problemă pe care trebuia s-o avem în vedere mai demult, şi anume *cât trebuie să achite clientul la fiecare împrumut (închirierea a una, două, trei sau patru casete)* ? În acest scop apelăm la atributul `ValoareÎnchir` (de la valoare închiriere):

(26) `IdÎmprumut` ⟶ `ValoareÎnchir`

În fine, ultima regulă, potrivit căreia *pierderea sau distrugerea unei casete antrenează o amendă care reprezintă dublul preţului casetei*, are o dublă implicaţie: pe de o parte, regretabilul eveniment trebuie consemnat la (un soi de) restituire, iar atributul `Penalizare` va conţine acum dublul contravalorii casetei; pe de altă parte, trebuie să existe cumva în baza de date o indicaţie precum că respectiva casetă nu mai poate fi închiriată, adică este pierdută sau distrusă. Soluţia ştergerii înregistrării corespunzătoare din tabela CASETE nu este una prea ingenioasă, întrucât ar trebui să pierdem toţi „copiii" acestei înregistrări, deci inclusiv de câte ori şi cui a fost împrumutată, şi, astfel, informaţiile statistice privind filmele, clienţii şi împrumuturile vor fi serios afectate. Cel mai nimerit pare să recurgem la un atribut numit `StareCasetă`, care să aibă o valoare implicită *OK* (sau nulă, deşi nu suntem fani NULL-işti) şi, dacă este cazul, *Pierdută*, *Distrusă*, ba chiar, dacă tot îl avem, putem consemna şi situaţiile în care caseta a fost scoasă din uz „de moarte bună" (*Casată*) sau se apropie de această stare (*Uzată*):

(27) `IdCasetă` ⟶ `StareCasetă`

Nu este însă suficient să cunoaştem starea casetei, ci şi de la ce împrumut i se trage pierderea sau distrugerea. Aşa că apelăm la un atribut special, `StareLaRestituire`, care indică halul în care un client ne-a restituit caseta (să fim înţeleşi: e vorba de halul casetei, nu al clientului!) şi care depinde funcţional astfel:

(28) (`IdÎmprumut`, `IdCasetă`) ⟶ `StareLaRestituire`

Ca un alt artificiu propus, ne imaginăm că, în ton cu vremurile, clienţii vor putea să-şi rezerve/comande casete pe Web, iar centrul să aibă un serviciu de furnizare a casetelor la domiciliu (de tipul *pizza delivery*) sau, în cel mai rău caz, datorită dimensiunii impresionante a centrului (dă, Doamne!), vor fi amplasate trei-patru terminale într-un colţ prin care clientul poate să caute filmele după actori, regizori, subiect etc. Tocmai pentru a evita execuţia unei interogări care să implice tabelele CASETE, ÎMPRUMUTURI şi CASETE_ÎMPRUMUTATE prin care să se afle dacă o casetă este disponibilă sau împrumutată la un moment dat, se poate recurge la un alt atribut redundant, numit `Disponibilitate`, actualizabil automat prin declanşatoarele tabelei CASETE_ÎMPRUMUTATE:

(29) `IdCasetă` ⟶ `Disponibilitate`

Apropo, ne propusesem şi un mic mecanism de *rating* al filmelor de către clienţi, deşi, din câte am văzut pe site-urile româneşti dedicate vânzărilor (legale) de carte şi CD-uri, sistemul nu prea a prins la noi. La modul cel mai simplu, putem alege o scală de punctare de la 0 la 10, de genul celei practicate de Laurenţiu Brătan în revista *22* (deşi acesta foloseşte o scală de la 0 la 5), atributul `Punctaj` fiind „implicat" în dependenţa :

(30) `(IdFilm, CNPClient)` ⟶ `Punctaj`

Tot frământându-ne mintea cu mecanismul de căutare a informaţiilor despre filme, realizăm că, la un moment dat, este posibil ca unii clienţi să dorească a viziona filme în care joacă actori spanioli, sau filme regizate de cineaşti născuţi în perioada 1930-1940 sau site-ul/pagina Web dedicată unei anumite personalităţi cinematografice. Putem vorbi de generalizare : actorii, regizorii şi scenariştii sunt toţi cineaşti, ca să nu amintim numeroşi actori care sunt şi regizori, sau regizori ce sunt şi producători şi scenarişti. Aşa că am putea introduce o serie de atribute pentru identificarea şi caracterizarea oricărui actor/scenarist/producător/regizor : `IdCineast`, `NumeCineast`, `DataNaşterii`, `DataMorţii`, `Naţionalitate`, `PaginăWeb`. Fără a schimba numele celor patru atribute (ar fi fost mai nimerite titulaturile `IdScenarist`, `IdProducător`, `IdRegizor` şi `IdActor`), reprezentăm cele patru dependenţe de incluziune `Scenarist` ⊆ `IdCineast`, `Producător` ⊆ `IdCineast`, `Regizor` ⊆ `IdCineast`, `Actor` ⊆ `IdCineast`.

Punând cap la cap tot ceea ce am discutat despre legăturile semantice dintre atributele bazei de date, obţinem reprezentarea sub formă de graf din figura 7.12. Pentru un plus de lizibilitate, am grupat toate destinaţiile surselor `IdFilm`, `IdCineast`, `IdCasetă` şi `CNPClient`.

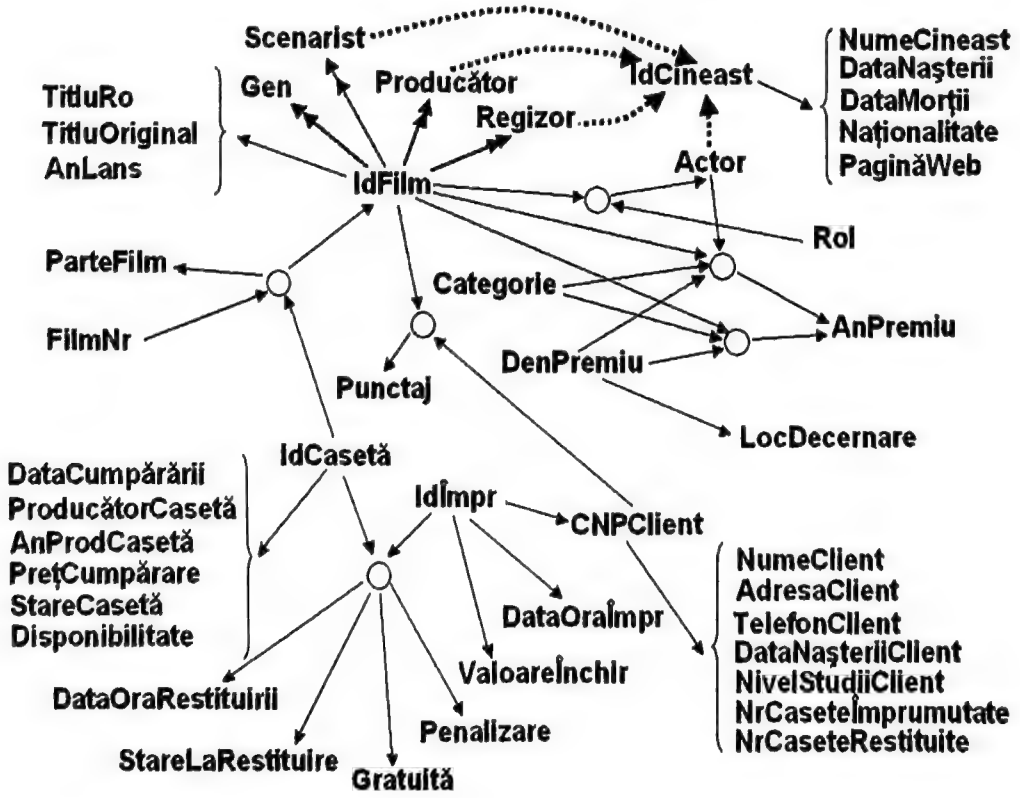


Figura 7.12. Graful DF pentru BD a centrului de închiriere – versiunea 2.0

Pe baza grafului, schema finală a bazei de date este următoarea:

PREMII_DENUMIRI {DenPremiu, LocDecernare}
FILME {IdFilm, TitluOriginal, TitluRO, AnLans}
CINEAŞTI {IdCineast, NumeCineast, DataNaşterii, DataMorţii, Naţionalitate, PaginăWeb}
DISTRIBUŢIE {IdFilm, Rol, Actor*}
PREMII_FILME {IdFilm, DenPremiu, Categorie, AnPremiu}
PREMII_INTERPRETARE {IdFilm, Actor*, DenPremiu, Categorie, AnPremiu}
PRODUCĂTORI {IdFilm, Producător*}
REGIZORI {IdFilm, Regizor*}
SCENARIŞTI {IdFilm, Scenarist*}
FILME_GENURI {IdFilm, Gen}
CASETE {IdCasetă, DataCumpărării, ProducătorCasetă, AnProdCasetă, PreţCumpărare, StareCasetă, Disponibilitate}
CASETE_FILME {IdCasetă, FilmNr, IdFilm, ParteFilm}
CLIENŢI {CNPClient, NumeClient, AdresaClient, TelefonClient, DataNaşteriiClient, NivelStudiiClient, NrCaseteÎmprumutate, NrCaseteRestituite}
ÎMPRUMUTURI {IdÎmpr, DataOraÎmpr, CNPClient, ValoareÎnchir}
CASETE_ÎMPRUMUTATE {IdÎmpr, IdCasetă, DataOraRestituirii, Gratuită, Penalizare, StareLaRestituire }
APRECIERI_FILME {IdFilm, CNPClient, Punctaj}

Atributele marcate cu asterisc sunt chei străine, părintele comun fiind câmpul `IdCineast` din tabela CINEAŞTI.

Fiind vorba de un caz practic pe care dorim să-l ducem cât mai aproape de „pânzele albe", vom urma promisiunea din deschiderea capitolului, discutând şi câteva chestiuni de implementare a schemei propuse. Începem cu crearea tabelelor şi definirea restricţiilor de nenulitate, cheilor primare, restricţiilor de integritate referenţială, precum şi a regulilor de validare la nivel de atribut şi înregistrare – vezi *listing*-ul 7.27. Atributele pentru care au fost definite restricţii sunt:

- `AnLans` din FILME (valori mai mari decât 1900);
- `AnPremiu` din PREMII_FILME şi PREMII_INTERPRETARE (valori mai mari decât 1920);
- `DataCumpărării` din CASETE (valori mai mari de 1 ianuarie 1998);
- `AnProdCaseta` din CASETE (valori mai mari decât 1980);
- `StareCaseta` din CASETE – valori limitate la lista: *Ok*, *Pierdută*, *Distrusă*, *Casată* şi *Uzată*;
- `Disponibilitate` din CASETE – numai valorile *D* (da) şi *N* (nu);
- `FilmNr` din CASETE_FILME: deoarece pe o casetă nu pot încăpea mai mult de 30 de clipuri/scurtmetraje, valorile acestui atribut trebuie să se încadreze în plaja 1-30;

- `ParteFilm` din CASETE_FILME: deoarece un film poate fi „rupt" în maximum 30 de părţi (nu credem ca vreun centru să achiziţioneze pe casete *Tânăr şi neliniştit*!), valorile acestui atribut trebuie să se încadreze în plaja 1-30;
- `NrCaseteImprumutate` şi `NrCaseteRestituite` din CLIENŢI – valorile trebuie să fie întregi şi pozitive;
- `StareLaRestituire` din CASETE_ÎMPRUMUTATE – valori limitate la lista: *Ok*, *Pierdută*, *Distrusă*;
- `Gratuită` din CASETE_ÎMPRUMUTATE – numai valorile *D* (da) şi *N* (nu);
- `Punctaj` din APRECIERI_FILME trebuie să se încadreze între 0 şi 10.

Listing 7.27. Scriptul Oracle de creare a tabelelor

```
-- creare BD centru de inchirieri VIDEO
--
DROP TABLE aprecieri_filme ;
DROP TABLE casete_imprumutate ;
DROP TABLE imprumuturi ;
DROP TABLE clienti ;
DROP TABLE casete_filme ;
DROP TABLE casete ;
DROP TABLE premii_interpretare ;
DROP TABLE premii_filme ;
DROP TABLE premii_denumiri ;
DROP TABLE scenaristi ;
DROP TABLE regizori ;
DROP TABLE producatori ;
DROP TABLE distributie ;
DROP TABLE filme_genuri ;
DROP TABLE cineasti ;
DROP TABLE filme ;

CREATE TABLE cineasti (
    IdCineast NUMBER(6) NOT NULL PRIMARY KEY,
    NumeCineast VARCHAR2(40) NOT NULL,
    DataNasterii DATE,
    DataMortii DATE,
    Naţionalitate VARCHAR2(30),
    PaginaWeb VARCHAR2(200),
    CHECK (NVL(DataNasterii, DATE'1970-01-01') <
        NVL2(DataMortii, DataNasterii, DATE'1970-01-02'))
    );

CREATE TABLE filme (
    IdFilm NUMBER(10) NOT NULL PRIMARY KEY,
    TitluOriginal VARCHAR2(100),
    TitluRO VARCHAR2(100) NOT NULL,
    AnLans NUMBER(4) CHECK (AnLans > 1900)
    ) ;
```

```
CREATE TABLE filme_genuri (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    Gen VARCHAR2(25) NOT NULL,
    PRIMARY KEY (IdFilm, Gen)
    ) ;

CREATE TABLE distributie (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    Rol VARCHAR2(30) NOT NULL,
    Actor NUMBER(6) REFERENCES cineasti (IdCineast),
    PRIMARY KEY (IdFilm, Rol)
    );

CREATE TABLE producatori (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    Producator NUMBER(6) NOT NULL REFERENCES cineasti (IdCineast),
    PRIMARY KEY (IdFilm, Producator)
    );

CREATE TABLE regizori (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    Regizor NUMBER(6) NOT NULL REFERENCES cineasti (IdCineast),
    PRIMARY KEY (IdFilm, Regizor)
    ) ;

CREATE TABLE scenaristi (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    Scenarist NUMBER(6) NOT NULL REFERENCES cineasti (IdCineast),
    PRIMARY KEY (IdFilm, Scenarist)
    );

CREATE TABLE premii_denumiri (
    DenPremiu VARCHAR2(40) NOT NULL PRIMARY KEY,
    LocDecernare VARCHAR2(50)
    );

CREATE TABLE premii_filme (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    DenPremiu VARCHAR2(40) NOT NULL REFERENCES premii_denumiri (DenPremiu),
    Categorie VARCHAR2(30) NOT NULL,
    AnPremiu NUMBER(4) CHECK (Anpremiu > 1920),
    PRIMARY KEY (IdFilm, DenPremiu, Categorie)
    );

CREATE TABLE premii_interpretare (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    Actor NUMBER(6) REFERENCES cineasti (IdCineast),
    DenPremiu VARCHAR2(40) NOT NULL REFERENCES premii_denumiri (DenPremiu),
    Categorie VARCHAR2(30) NOT NULL,
    AnPremiu NUMBER(4) CHECK (AnPremiu > 1920),
```

```
    PRIMARY KEY (IdFilm, Actor, DenPremiu, Categorie)
    ) ;

CREATE TABLE casete (
    IdCaseta NUMBER(12) NOT NULL PRIMARY KEY,
    DataCumpararii DATE CHECK (DataCumpararii > DATE'1998-01-01'),
    ProducătorCaseta VARCHAR2(30),
    AnProdCaseta NUMBER(4) DEFAULT EXTRACT (YEAR FROM CURRENT_DATE)
        CHECK (AnProdCaseta > 1980),
    PreţCumparare NUMBER(8),
    StareCaseta VARCHAR2(20) DEFAULT 'Ok' NOT NULL
        CHECK (StareCaseta IN ('Ok', 'Pierduta', 'Distrusa', 'Casata', 'Uzata')),
    Disponibilitate CHAR(1) DEFAULT 'D' NOT NULL
        CHECK (Disponibilitate IN ('D','N'))
    );

CREATE TABLE casete_filme (
    IdCaseta NUMBER(12) NOT NULL REFERENCES casete (Idcaseta),
    FilmNr NUMBER(2) DEFAULT 1 NOT NULL
        CHECK (FilmNr BETWEEN 1 AND 30),
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    ParteFilm NUMBER(2) DEFAULT 1 NOT NULL
        CHECK (ParteFilm BETWEEN 1 AND 30),
    PRIMARY KEY (IdCaseta, FilmNr)
    ) ;

CREATE TABLE clienti (
    CNPClient NUMBER(13) NOT NULL PRIMARY KEY,
    NumeClient VARCHAR2(45) NOT NULL,
    AdresaClient VARCHAR2(100),
    TelefonClient VARCHAR2(25),
    DataNasteriiClient DATE,
    NivelStudiiClient VARCHAR2(15) DEFAULT 'Medii',
    NrCaseteImprumutate NUMBER(6) DEFAULT 0
        CHECK (NrCaseteImprumutate >= 0),
    NrCaseteRestituite NUMBER(6) DEFAULT 0
        CHECK (NrCaseteRestituite >= 0),
    CHECK (NrCaseteImprumutate - NrCaseteRestituite BETWEEN 0 AND 4)
    );

CREATE TABLE imprumuturi (
    IdImpr NUMBER(15) NOT NULL PRIMARY KEY,
    DataOraImpr DATE DEFAULT CURRENT_DATE,
    CNPClient NUMBER(13) NOT NULL REFERENCES clienti(CNPClient),
    ValoareInchir NUMBER(7) DEFAULT 0 NOT NULL ;
    );

CREATE TABLE casete_imprumutate (
    IdImpr NUMBER(15) NOT NULL REFERENCES imprumuturi (IdImpr),
    IdCaseta  NUMBER(12) NOT NULL REFERENCES casete (Idcaseta),
```

```
    DataOraRestituirii DATE,
    StareLaRestituire VARCHAR2(20) DEFAULT 'Ok'
        CHECK (StareLaRestituire IN ('Ok', 'Pierduta', 'Distrusa')),
    Gratuita CHAR(1) DEFAULT 'N' NOT NULL
        CHECK (Gratuita IN ('D','N')),
    Penalizare NUMBER(9) DEFAULT 0,
    PRIMARY KEY (IdImpr, IdCaseta)
    );

CREATE TABLE aprecieri_filme (
    IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
    CNPClient  NUMBER(13) NOT NULL REFERENCES clienti(CNPClient),
    Punctaj NUMBER(2) DEFAULT 0 NOT NULL
        CHECK (Punctaj BETWEEN 0 AND 10),
    PRIMARY KEY (IdFilm, CNPClient)
    );
```

Regulile de validare la nivel de înregistrare prezente în script privesc următoarele restricţii :

- în tabela **CINEAŞTI** : eventuala dată a decesului să succeadă datei de naştere ; întrucât, pe de o parte, pentru cineaştii în viaţă, `DataMorţii` este nulă, iar, pe de altă parte, pentru o serie de cineaşti nu se cunoaşte încă una sau ambele date, trebuie avute în vedere problemele ce pot apărea din „manevrarea" valorilor nule ; cea mai nimerită este, în acest context, funcţia NVL2[14] ;
- în **CLIENŢI**, având în vedere că nici un client nu poate închiria, la orice moment dat, mai mult de patru casete, regula este una simplă : `NrCaseteImprumutate-NrCaseteRestituite BETWEEN 0 AND 4` ; mai dificil va fi mecanismul de asigurare a corectitudinii celor două atribute.

Continuăm implementarea în Oracle a regulilor stabilite, rezolvând problema cheilor surogat, care prespune folosirea secvenţelor şi declanşatoarelor de inserare, după modelul din *listing*-ul 7.28. După cum spuneam şi cu alte ocazii, majoritatea SGBD-urilor disponibile permit declararea de atribute autoincrementate.

Listing 7.28. Secvenţe şi declanşatoare de inserare PL/SQL pentru generarea cheilor surogat

```
-- autoincrementarea atributului CINEASTI.IdCineast
DROP SEQUENCE seq_IdCineast ;
CREATE SEQUENCE seq_IdCineast START WITH 1
    MINVALUE 1 MAXVALUE 999999
    NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_cineasti_ins
    BEFORE INSERT ON cineasti FOR EACH ROW
```

14. Pentru câteva detalii despre funcţia NVL2, vezi şi [Fotache ş.a. 03], p. 190.

```
BEGIN
    SELECT seq_IdCineast.NextVal INTO :NEW.IdCineast FROM dual ;
END;
/

-- autoincrementarea atributului FILME.IdFilm
DROP SEQUENCE seq_IdFilm ;
CREATE SEQUENCE seq_IdFilm START WITH 1
    MINVALUE 1 MAXVALUE 9999999999
    NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_filme_ins
    BEFORE INSERT ON filme FOR EACH ROW
BEGIN
    SELECT seq_IdFilm.NextVal INTO :NEW.IdFilm FROM dual ;
END;
/

-- autoincrementarea atributului CASETE.IdCaseta
DROP SEQUENCE seq_IdCaseta ;
CREATE SEQUENCE seq_IdCaseta START WITH 10000000001
    MINVALUE 1 MAXVALUE 999999999999
    NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_casete_ins
    BEFORE INSERT ON casete FOR EACH ROW
BEGIN
    SELECT seq_IdCaseta.NextVal INTO :NEW.IdCaseta FROM dual ;
END;
/

-- autoincrementarea atributului IMPRUMUTURI.IdImpr
DROP SEQUENCE seq_IdImpr ;
CREATE SEQUENCE seq_IdImpr START WITH 1000000000001
    MINVALUE 1 MAXVALUE 99999999999999
    NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_imprumuturi_ins
    BEFORE INSERT ON imprumuturi FOR EACH ROW
BEGIN
    SELECT seq_IdImpr.NextVal INTO :NEW.IdImpr FROM dual ;
END;
/
```

Tabela CASETE_FILME conţine un caz special de atribut a cărui valoare trebuie gestionată automat. Astfel, atributul `FilmNr` trebuie iniţializat cu 1 la primul film de pe fiecare casetă, apoi trebuie incrementat cu 1 pentru orice alt film declarat ulterior ca fiind înregistrat pe caseta respectivă. Necazul este că, la ştergerea unui film de pe o casetă, trebuie reverificată ordinea filmelor pe caseta respectivă. De exemplu, dacă pe o casetă sunt 5 filme (scurtmetraje) şi se şterge linia care se referă la al treilea film

de pe casetă, atunci filmul al patrulea devine al treilea, iar al cincilea devine al patrulea.

Iar pentru ca delectarea să fie totală, trebuie să se interzică modificarea interactivă (prin UPDATE) atât a atributului `CASETE_FILME.FilmNr`, cât şi a atributelor `CASETE_FILME.IdCasetă` şi `CASETE_FILME.IdFilm`. Aceste ultime două atribute pot fi, eventual, modificate doar prin declanşatoarele de tip UPDATE CASCADE la actualizarea atributelor `CASETE.IdCasetă` şi `FILME.IdFilm`.

Iată în *listing*-ul 7.29 cele trei declanşatoare ale tabelei CASETE_FILME, împreună cu pachetul necesar declarării variabilelor globale.

Listing 7.29. Un pachet şi patru declanşatoare PL/SQL pentru gestionarea valorilor atributului FilmNr din CASETE_FILME

```
-------------------------------------------------------------
-- declansatorul de inserare
CREATE OR REPLACE TRIGGER trg_casete_filme_ins
    BEFORE INSERT ON casete_filme FOR EACH ROW
DECLARE
    v_filmnr casete_filme.FilmNr%TYPE ;
BEGIN
    SELECT MAX(FilmNr) INTO v_filmnr FROM casete_filme
    WHERE IdCaseta = :NEW.IdCaseta ;

    :NEW.FilmNr := NVL(v_filmnr,0) + 1 ;
END ;
/

-------------------------------------------------------------
-- pachetul cu variabile publice
CREATE OR REPLACE PACKAGE pac_centru AS
    -- variabila pentru semnalarea locului de actualizare
    --  in CASETE_FILME
    vp_trg_casete_filme_del BOOLEAN := FALSE ;
    vp_idcaseta casete.IdCaseta%TYPE ;
    vp_filmnr casete_filme.FilmNr%TYPE ;

    -- variabile pentru semnalarea modificarilor in cascada
    vp_trg_casete_upd BOOLEAN := FALSE ;
    vp_trg_filme_upd BOOLEAN := FALSE ;

END pac_centru ;
/

-------------------------------------------------------------
-- declansator de stergere LA NIVEL DE LINIE
CREATE OR REPLACE TRIGGER trg_casete_filme_del_after_row
    AFTER DELETE ON casete_filme FOR EACH ROW
DECLARE
    v_filmnrcrt casete_filme.FilmNr%TYPE ;
```

```
BEGIN
    pac_centru.vp_trg_casete_filme_del := TRUE ;
    pac_centru.vp_idcaseta := :OLD.IdCaseta ;
    pac_centru.vp_filmnr := :OLD.FilmNr ;
    pac_centru.vp_trg_casete_filme_del := TRUE ;
END;
/

-- declansatorul de stergere LA NIVEL DE COMANDA
CREATE OR REPLACE TRIGGER trg_casete_filme_del_stat
    AFTER DELETE ON casete_filme
DECLARE
    v_filmcrt casete_filme.FilmNr%TYPE ;

    CURSOR c_casete_filme (idcaseta_ casete.IdCaseta%TYPE,
                filmcrt_ casete_filme.FilmNr%TYPE)
        IS SELECT * FROM casete_filme
         WHERE IdCaseta = idcaseta_ AND FilmNr >= filmcrt_
            ORDER BY FilmNr ;
BEGIN
    pac_centru.vp_trg_casete_filme_del := TRUE ;
    v_filmcrt := pac_centru.vp_filmnr ;

    FOR rec_casete_filme IN c_casete_filme (pac_centru.vp_idcaseta,
            pac_centru.vp_filmnr) LOOP
        IF rec_casete_filme.FilmNr = v_filmcrt THEN
            UPDATE casete_filme
            SET FilmNr = 29
            WHERE IdCaseta=rec_casete_filme.IdCaseta AND
                FilmNr=rec_casete_filme.FilmNr ;
        ELSE
            UPDATE casete_filme
            SET FilmNr = v_filmcrt
            WHERE IdCaseta = rec_casete_filme.IdCaseta AND
                FilmNr=rec_casete_filme.FilmNr ;
            v_filmcrt := v_filmcrt + 1 ;
        END IF ;
    END LOOP;

    pac_centru.vp_trg_casete_filme_del := FALSE ;
END;
/

-----------------------------------------------------------------
-- declansator de modificare
CREATE OR REPLACE TRIGGER trg_casete_filme_upd
    BEFORE UPDATE ON casete_filme FOR EACH ROW
DECLARE
    v_filmnrcrt casete_filme.FilmNr%TYPE ;
BEGIN
```

```
    IF :NEW.IdCaseta <> :OLD.IdCaseta AND
            pac_centru.vp_trg_casete_upd = FALSE THEN
        RAISE_APPLICATION_ERROR(-20789,
            'Nu puteti modifica interactiv IdCaseta!');
    END IF;
    IF :NEW.IdFilm <> :OLD.IdFilm AND
            pac_centru.vp_trg_filme_upd = FALSE THEN
        RAISE_APPLICATION_ERROR(-20788,
            'Nu puteti modifica interactiv IdFilm!');
    END IF;
    IF :OLD.FilmNr <> :NEW.FilmNr AND
            pac_centru.vp_trg_casete_filme_del = FALSE THEN
        RAISE_APPLICATION_ERROR(-20787,
            'Nu puteti modifica interactiv FilmNr!');
    END IF;
END;
/
```

Nici n-am apucat bine să ne despărţim de tratarea cheilor surogat, că ne şi întoarcem la ele, de data aceasta punându-ne problema actualizării lor. În mod obişnuit, într-o restricţie referenţială modificarea valorii cheii primare (alternative) atrage după sine necesitatea modificării automate a valorilor tuturor cheilor străine ale acesteia. În standardele SQL şi în unele SGBD-uri există, la creare, opţiunea `ON UPDATE CASCADE`. Oracle însă este mai zgârcit în privinţa asta, aşa că operaţiunea de modificare în cascadă trebuie întreprinsă prin declanşatoarele de modificare a tabelei-părinte.

Dacă în cazul atributului `CLIENTI.CNPClient` nu se întrezăreşte nici o problemă, pentru tabelele ce folosesc chei surogat gestionate prin secvenţă apare un necaz. Să presupunem că în tabela CINEAŞTI ultima linie introdusă are valoarea `IdCineast` egală cu 344, iar secvenţa `SEQ_IDCINEAST` s-a oprit pe această „poziţie". În acest moment se lansează o comandă `UPDATE` prin care ID-ul 256 este făcut 345 :

```
UPDATE cineasti SET IdCineast = 345 WHERE IdCineast=256;
```

Necazul apare la următoarea comandă `INSERT` în CINEAŞTI, deoarece valoarea „secretată" de secvenţă se suprapune peste valoarea stabilită prin UPDATE, iar restricţia de cheie primară va fi încălcată. De aceea, în declanşatoarele de modificare în cascadă a atributelor cheilor surogat se va testa în prealabil dacă noua valoare depăşeşte limita actuală a secvenţei – vezi *listing*-ul 7.30.

Listing 7.30. Declanşatoarele pentru propagarea modificărilor cheilor primare în tabelele-copil

```
-------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_cineasti_upd
    AFTER UPDATE OF IdCineast ON cineasti FOR EACH ROW
DECLARE
```

```
    v_urmatorulId cineasti.IdCineast%TYPE ;
BEGIN
    SELECT last_number INTO v_urmatorulId
    FROM user_sequences
    WHERE sequence_name='SEQ_IDCINEAST' ;
    IF :NEW.IdCineast >= v_urmatorulId THEN
        RAISE_APPLICATION_ERROR(-20779,
        'Valoarea IdCineast depaseste valoarea curenta a secventeil');
    ELSE
        UPDATE distributie SET Actor=:NEW.IdCineast
            WHERE Actor=:OLD.IdCineast ;
        UPDATE premii_interpretare SET Actor=:NEW.IdCineast
            WHERE Actor=:OLD.IdCineast ;
        UPDATE producatori SET Producator=:NEW.IdCineast
            WHERE Producator=:OLD.IdCineast ;
        UPDATE regizori SET Regizor=:NEW.IdCineast
            WHERE Regizor=:OLD.IdCineast ;
        UPDATE scenaristi SET Scenarist=:NEW.IdCineast
            WHERE Scenarist=:OLD.IdCineast ;
    END IF ;
END ;
/

-------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_filme_upd
    AFTER UPDATE OF IdFilm ON filme FOR EACH ROW
DECLARE
    v_urmatorulId filme.IdFilm%TYPE ;
BEGIN
    SELECT last_number INTO v_urmatorulId
    FROM user_sequences
    WHERE sequence_name='SEQ_IDFILM' ;
    IF :NEW.IdFilm >= v_urmatorulId THEN
        RAISE_APPLICATION_ERROR(-20778,
            'Valoarea IdFilm depaseste valoarea curenta a secventeil');
    ELSE
        pac_centru.vp_trg_filme_upd := TRUE ;
        UPDATE distributie SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE distributie SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE premii_filme SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE premii_interpretare SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE producatori SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE regizori SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE scenaristi SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE filme_genuri SET IdFilm=:NEW.IdFilm
```

```
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE casete_filme SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        UPDATE aprecieri_filme SET IdFilm=:NEW.IdFilm
            WHERE IdFilm=:OLD.IdFilm ;
        pac_centru.vp_trg_filme_upd := FALSE ;
    END IF ;
END ;
/

-----------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_casete_upd
    AFTER UPDATE OF IdCaseta ON casete FOR EACH ROW
DECLARE
    v_urmatorulId casete.IdCaseta%TYPE ;
BEGIN
    SELECT last_number INTO v_urmatorulId
    FROM user_sequences
    WHERE sequence_name='SEQ_IDCASETA' ;
    IF :NEW.Idcaseta >= v_urmatorulId THEN
        RAISE_APPLICATION_ERROR(-20778,
        'Valoarea IdCaseta depaseste valoarea curenta a secventei!');
    ELSE
        pac_centru.vp_trg_casete_upd := TRUE ;
        UPDATE casete_filme SET IdCaseta=:NEW.IdCaseta
            WHERE Idcaseta=:OLD.Idcaseta ;
        UPDATE casete_imprumutate SET IdCaseta=:NEW.IdCaseta
            WHERE Idcaseta=:OLD.Idcaseta ;
        pac_centru.vp_trg_casete_upd := FALSE ;
    END IF ;
END ;
/

-----------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_clienti_upd
    AFTER UPDATE OF CNPClient ON clienti FOR EACH ROW
BEGIN
    UPDATE imprumuturi SET CNPClient=:NEW.CNPClient
        WHERE CNPClient=:OLD.CNPClient ;
    UPDATE aprecieri_filme SET CNPClient=:NEW.CNPClient
        WHERE CNPClient=:OLD.CNPClient ;
END ;
/

-----------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_imprumuturi_upd
    AFTER UPDATE OF IdImpr ON imprumuturi FOR EACH ROW
DECLARE
    v_urmatorulId imprumuturi.IdImpr%TYPE ;
BEGIN
    SELECT last_number INTO v_urmatorulId
    FROM user_sequences
```

```
    WHERE sequence_name='SEQ_IDIMPR' ;
    IF :NEW.IdImpr >= v_urmatorulid THEN
        RAISE_APPLICATION_ERROR(-20778,
            'Valoarea IdImpr depaseste valoarea curenta a secventeI');
    ELSE
        UPDATE casete_imprumutate SET IdImpr=:NEW.IdImpr
            WHERE IdImpr=:OLD.IdImpr ;
    END IF ;
END ;
/
```

Putem bifa liniştiţi acum problemele ce ţin de restricţiile referenţiale din bază. Ne apropiem uşor de miezul problemei, anume de mecanismul de actualizare a atributelor legate de casetele împrumutate (închiriate) şi restituirea acestora, inclusiv regulile formulate cu privire la gratuităţi şi penalizări. Mecanismul de care vorbim se „învârte" în jurul declanşatoarelor tabelei CASETE_ÎMPRMUTATE. Din raţiuni de spaţiu, ne vom opri doar la declanşatorul de inserare în această tabelă – vezi *listing*-ul 7.31 – prefaţat de antetul şi specificaţiile pachetului PAC_CENTRU. După cum se poate observa, în pachet au fost introduse o serie de funcţii:

- F_NRCASETEÎNCHIRIATE1 – furnizează numărul de casete închiriate de un client dat;
- F_NRCASETEÎNCHIRIATE2 – furnizează numărul de casete închiriate de un client, cunoscând însă doar identificatorul împrumutului;
- F_CNPCLIENT – returnează codul numeric personal al unui client, cunoscând identificatorul împrumutului;
- F_CHIRIEZI – calculează tariful zilnic de închiriere a unei casete (pe baza anului în care a fost produsă);
- F_PREŢCASETĂ – returnează preţul la care a fost cumpărată caseta;
- F_DATAÎMPR – returnează data (şi ora) la care a avut loc un împrumut;
- F_CASETADISPONIBILĂ – returnează TRUE în cazul în care caseta nu este împrumutată şi FALSE în caz contrar.

Declararea acestor funcţii are ca scop atât micşorarea dimensiunii declanşatoarelor tabelei şi, implicit, o mai buna lizibilitate şi viteză de execuţie, cât şi modularizarea aplicaţiei, întrucât acestea vor fi necesare şi în celelalte (cel puţin) două declanşatoare – de modificare şi ştergere – pe care nu le vom mai discuta.

Listing 7.31. Declanşatorul de inserare în CASETE_ÎMPRUMUTATE, precedat de pachetul PAC_CENTRU

```
-------------------------------------------------------------------
-- pachetul cu variabile publice se intoarce
CREATE OR REPLACE PACKAGE pac_centru AS
    -- variabila pentru semnalizarea locului de actualizare
    --  in CASETE_FILME
    vp_trg_casete_filme_del BOOLEAN := FALSE ;
```

```
    vp_idcaseta casete.IdCaseta%TYPE ;
    vp_filmnr casete_filme.FilmNr%TYPE ;

    -- variabile pentru semnalizarea modificarilor in cascada
    vp_trg_casete_upd BOOLEAN := FALSE ;
    vp_trg_filme_upd BOOLEAN := FALSE ;

    -- functii
    FUNCTION f_nrcaseteinchiriate1
        (cnpclient_ clienti.CNPClient%TYPE)
        RETURN clienti.NrCaseteImprumutate%TYPE;
    FUNCTION f_nrcaseteinchiriate2 (idimpr_ imprumuturi.IdImpr%TYPE)
        RETURN clienti.NrCaseteImprumutate%TYPE;
    FUNCTION f_cnpclient (idimpr_ imprumuturi.IdImpr%TYPE)
        RETURN clienti.CNPClient%TYPE;
    FUNCTION f_chiriezi (idcaseta_ casete.Idcaseta%TYPE)
        RETURN NUMBER ;
    FUNCTION f_pretcaseta (idcaseta_ casete.IdCaseta%TYPE)
        RETURN casete.PretCumparare%TYPE ;
    FUNCTION f_dataimpr (idimpr_ casete_imprumutate.IdImpr%TYPE)
        RETURN imprumuturi.DataOraImpr%TYPE ;
    FUNCTION f_casetadisponibila (idcaseta_ casete.IdCaseta%TYPE)
        RETURN BOOLEAN ;

END pac_centru ;
/
-----------------------------------------------
-- corpul pachetului
CREATE OR REPLACE PACKAGE BODY pac_centru AS
    FUNCTION f_nrcaseteinchiriate1 (
        cnpclient_ clienti.CNPClient%TYPE)
        RETURN clienti.NrCaseteImprumutate%TYPE
    IS
        V_nrci clienti.NrCaseteImprumutate%TYPE ;
    BEGIN
        SELECT NrCaseteImprumutate INTO v_nrci
        FROM clienti
        WHERE CNPClient = cnpclient_ ;
        RETURN v_nrci ;
    END f_nrcaseteinchiriate1 ;
    ------------------------
    FUNCTION f_nrcaseteinchiriate2 (idimpr_ imprumuturi.IdImpr%TYPE)
        RETURN clienti.NrCaseteImprumutate%TYPE
    IS
        V_nrci clienti.NrCaseteImprumutate%TYPE ;
    BEGIN
        SELECT NrCaseteImprumutate INTO v_nrci
        FROM clienti
        WHERE CNPClient = f_cnpclient(idimpr_) ;
        RETURN v_nrci ;
```

```
END f_nrcaseteInchiriate2  ;
-------------------------
FUNCTION f_cnpclient (idimpr_ imprumuturi.idimpr%TYPE)
   RETURN clienti.CNPClient%TYPE
IS
   v_cnpclient clienti.CNPClient%TYPE ;
BEGIN
   SELECT CNPClient INTO v_cnpclient
   FROM imprumuturi
   WHERE idimpr = idimpr_ ;
   RETURN v_cnpclient ;
END f_cnpclient ;
-------------------------
FUNCTION f_chiriezi (idcaseta_ casete.idcaseta%TYPE)
   RETURN NUMBER
IS
   v_anprodcaseta casete.AnProdCaseta%TYPE ;
BEGIN
   SELECT NVL(AnProdCaseta, EXTRACT (YEAR FROM SYSDATE))
   INTO v_anprodcaseta
   FROM casete WHERE idcaseta=idcaseta_ ;
   CASE
   WHEN EXTRACT (YEAR FROM SYSDATE) - v_anprodcaseta
         BETWEEN 0 AND 1 THEN
      RETURN 50000 ;
   WHEN EXTRACT (YEAR FROM SYSDATE) - v_anprodcaseta
         BETWEEN 2 AND 3 THEN
      RETURN 40000  ;
   ELSE
      RETURN 30000 ;
   END CASE ;
END f_chiriezi ;
-------------------------
FUNCTION f_pretcaseta (idcaseta_ casete.idCaseta%TYPE)
   RETURN casete.PretCumparare%TYPE
IS
   v_pret casete.PretCumparare%TYPE ;
BEGIN
   SELECT PretCumparare INTO v_pret FROM casete
   WHERE idcaseta=idcaseta_ ;
   RETURN v_pret ;
END f_pretcaseta ;
-------------------------
FUNCTION f_dataimpr (idimpr_ casete_imprumutate.idImpr%TYPE)
   RETURN imprumuturi.DataOraImpr%TYPE
IS
   v_data imprumuturi.DataOraImpr%TYPE ;
BEGIN
   SELECT DataOraImpr INTO v_data
   FROM imprumuturi WHERE idImpr = idimpr_ ;
   RETURN v_data ;
```

```
    END f_dataImpr ;
    ----------------------------
    FUNCTION f_casetaDisponibila (idcaseta_ casete.idCaseta%TYPE)
        RETURN BOOLEAN
    IS
        v_disponibilitate CHAR(1);
    BEGIN
        SELECT Disponibilitate INTO v_disponibilitate
        FROM casete WHERE idCaseta=idcaseta_ ;
        IF v_disponibilitate='D' THEN
            RETURN TRUE;
        ELSE
            RETURN FALSE ;
        END IF ;
    END f_casetaDisponibila ;

END pac_centru ;
/

-----------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_casete_imprumutate_ins
    BEFORE INSERT ON casete_imprumutate FOR EACH ROW

DECLARE
    v_rest NUMBER(2) ; -- restul impartirii la 11 (pentru caseta
                       --     inchiriata gratuit)
BEGIN
    -- se inregistreaza imprumutul unei casete

    -- se creste numarul casetelor imprumutate clientului respectiv
    UPDATE clienti
    SET NrCaseteImprumutate = NrCaseteImprumutate + 1
    WHERE CNPClient = pac_centru.f_cnpclient(:NEW.idImpr) ;

    v_rest := MOD(pac_centru.f_nrcaseteInchiriate2(:NEW.idImpr),11) ;

    IF v_rest = 0 THEN
        :NEW.Gratuita := 'D' ;
    ELSE
        :NEW.Gratuita := 'N' ;
        UPDATE imprumuturi SET ValoareInchir = ValoareInchir +
            pac_centru.f_chiriezi(:NEW.idCaseta)
        WHERE idImpr = :NEW.idImpr ;
    END IF ;

    IF :NEW.DataOraRestituirii IS NULL THEN
        -- se verifica disponibilitatea casetei
        IF pac_centru.f_casetadisponibila(:NEW.idCaseta) = FALSE THEN
            RAISE_APPLICATION_ERROR(-20678, 'Caseta indisponibila') ;
        END IF ;
```

```
        -- caseta se declara indisponibila (pana la returnare)
        UPDATE casete SET Disponibilitate='N'
            WHERE IdCaseta = :NEW.IdCaseta ;
        :NEW.penalizare := 0 ;
    ELSE
        -- caseta e deja returnata (se opereaza simultan
        --  inchirierea si returnarea)
        UPDATE casete SET Disponibilitate='D'
            WHERE IdCaseta = :NEW.IdCaseta ;

        -- se incrementeaza numarul casetelor returnate
        --  de clientul respectiv
        UPDATE clienti
        SET NrCaseteRestituite = NrCaseteRestituite + 1
        WHERE CNPClient IN
            (SELECT CNPClient FROM imprumuturi
            WHERE IdImpr=:NEW.IdImpr) ;

        -- se verifica intarzierile la returnare (mai mult de 2 zile)
        IF :NEW.DataOraRestituirii -
                pac_centru.f_dataimpr (:NEW.IdImpr) > 2 THEN
            :NEW.penalizare := 0.75 *
                pac_centru.f_chiriezi(:NEW.IdCaseta) *
            (:NEW.DataOraRestituirii -
                pac_centru.f_dataimpr (:NEW.IdImpr) - 2)  ;
        ELSE
            :NEW.penalizare := 0 ;
        END IF ;
    END IF ;

    IF :NEW.StareLaRestituire IN ('Pierduta', 'Distrusa') THEN
        UPDATE casete
        SET StareCaseta = :NEW.StareLaRestituire
        WHERE IdCaseta=:NEW.Idcaseta ;
        :NEW.Penalizare := :NEW.Penalizare +
            2 * pac_centru.f_pretcaseta(:NEW.IdCaseta) ;
    END IF ;

END ;
/
```

În privinţa declanşatorului propriu-zis, pornim de la premisa că inserarea se poate referi atât la operarea *„online”* a închirierii unei casete, dar şi la operarea unui împrumut mai vechi, pentru care se înregistrează atât închirierea, cât şi restituirea. Logica declanşatorului este următoarea:

- se incrementează numărul casetelor împrumutate de clientul respectiv;
- se calculează dacă numărul respectivei casete împrumutate clientului este multiplu de 11, caz în care închirierea sa este gratuită;
- dacă nu avem cazul de gratuitate de mai sus, se adaugă chiria casetei curente la valoarea pe care trebuie să o plătească clientul pentru împrumutul din care face parte şi caseta curentă;

- dacă se operează împrumutul unei casete ce nu a fost restituită până în momentul inserării (acesta este, de fapt, cazul cel mai frecvent), se verifică dacă se poate închiria caseta (dacă este disponibilă); după operarea împrumutului, caseta se declară indisponibilă (până la restituirea sa);
- dacă inserarea priveşte o casetă împrumutată şi restituită, atunci se incrementează numărul de casete restituite de client, şi apoi se trece la calcularea eventualelor penalităţi de întârziere sau a eventualei penalităţi de pierdere sau distrugere a casetei.

Din păcate, mai este destul de mult de lucru. Mai trebuie create:
- declanşatoarele de ştergere şi modificare pentru CASETE_ÎMPRUMUTATE;
- declanşatorul de modificare pentru CASETE, întrucât o casetă poate fi pierdută sau distrusă şi de personalul propriu sau din pricina unui incendiu, unei inundaţii etc.; pe de altă parte, pentru atributele calculate nu trebuie permisă modificarea interactivă (altfel decât prin declanşatoare);
- declanşatorul de modificare pentru CLIENŢI şi ÎMPRUMUTURI: pentru atributele calculate nu trebuie permisă modificarea interactivă (altfel decât prin declanşatoare).

Pentru detalii privitoare la aceste operaţiuni, puteţi consulta şi lucrarea [Fotache ş.a. 03].

CAPITOLUL 8

Generalizări, ierarhii, grafuri, echivalenţe

Chiar dacă am tratat şi dependenţele de incluziune, pe nedrept omise de la normalizare de majoritatea lucrărilor dedicate proiectării bazelor de date, în capitolul 7 s-a simţit o boare de postnormalizare, întrucât am pornit de la o schemă normalizată pe care am garnisit-o cu elemente ce ţin de implementarea unor restricţii mai sofisticate, tolerând chiar o doză de redundanţă în numele restricţiilor respective. În acest capitol continuăm cu probleme ce nu sunt legate, de obicei, de teoria şi practica normalizării, dar care îşi au importanţa lor în elaborarea unei scheme cât mai aproape de dezideratele aplicaţiilor din lumea afacerilor sau colaterale. Mai precis, ne vom ocupa, pe rând, de problemele generalizării/specializării, ierarhiilor, grafurilor şi relaţiilor de echivalenţă.

8.1. Generalizare şi specializare

Problema generalizării în bazele de date relaţionale a fost abordată înaintea definirii dependenţelor de incluziune. Astfel, în două articole publicate în 1977, John şi Diane Smith delimitează două tipuri fundamentale de abstractizare: agregarea şi generalizarea[1]. Agregarea transformă o relaţie dintre obiecte într-un obiect superior, cei doi autori propunând un nou tip de date care respectă anumite criterii de „bună definire", numit *agregat*[2]. Ei dau ca exemplu relaţia dintre o persoană, un hotel şi o dată calendaristică ce poate fi abstractizată ca un nou obiect - *rezervare*. Pe noi însă ne interesează cu deosebire conceptul de generalizare care reprezintă genul de abstractizare prin care clase de obiecte sunt transformate într-un obiect „mai general". Altfel spus, o generalizare este o abstractizare care permite unei clase de obiecte individuale să fie gândită, la modul general, ca un obiect în sine[3].

Generalizare şi specializare simplă

Vom discuta, pentru început, unul dintre exemplele celor doi autori, pentru că structura la care ajung aceştia este una discutabilă. Pentru o companie transnaţională,

1. [Smith & Smith 77-1], [Smith & Smith 77-2].
2. [Smith & Smith 77-1].
3. [Smith & Smith 77-2].

agenţie guvernamentală (secretă sau nu) sau minister al apărării se pune problema evidenţierii tuturor vehiculelor din dotare. Datorită complexităţii acesteia, se poate apela la o ierarhie precum cea din figura 8.1.

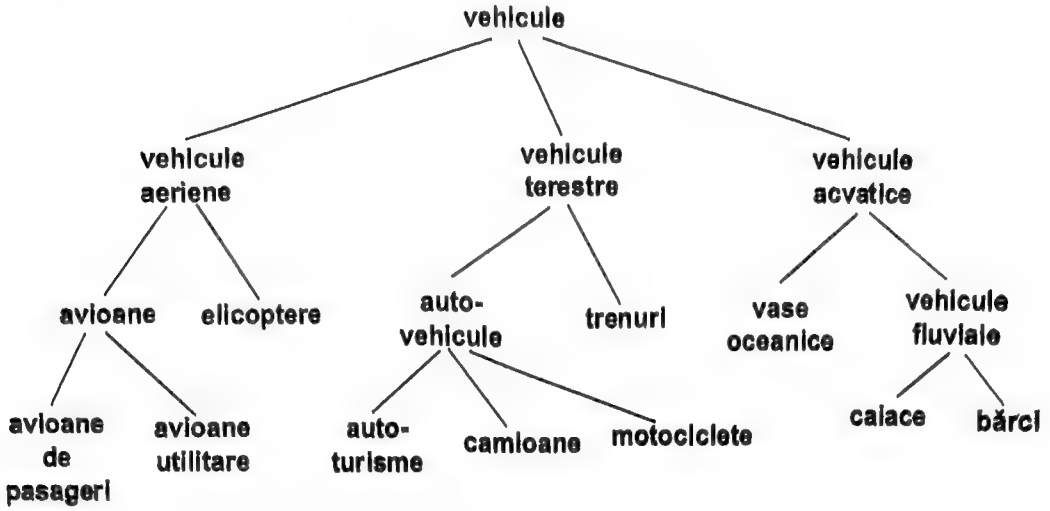


Figura 8.1. O ierarhie de vehicule (adaptare după [Smith&Smith77-2])

Avioanele de pasageri şi cele utilitare pot fi generalizate, definindu-se categoria *avioane*, iar noţiunile *autoturisme*, *camioane* şi *motociclete* pot fi generalizate în categoria *autovehicule*. O parte dintre atribute sunt comune tuturor categoriilor de vehicule : identificatorul vehiculului, denumirea, producătorul, greutatea şi preţul. Acestea sunt definite la nivelul obiectului „rădăcină" – *vehicule* –, fiind „moştenite" de toate obiectele subordonate. O altă parte însă este specifică fiecărei categorii de obiecte :

- pentru vehicule aeriene : înălţimea maximă de zbor ; viteza maximă ; numărul de ore de autonomie de zbor ; capacitatea rezervorului, tipul de carburant ;
- pentru elicoptere : tipul de rotor ; numărul paletelor ce alcătuiesc elicea principală ;
- pentru avioane : unghiul maxim de decolare ; lungimea minimă a pistei pentru decolare/aterizare ;
- pentru avioane de pasageri : numărul de pasageri ;
- pentru avioane utilitare : sarcina utilă ;
- pentru vehicule terestre : viteza maximă, tipul de carburant ;
- pentru autovehicule : numărul de roţi ; capacitatea cilindrică ; numărul de cai putere ; consumul mixt la 100 Km ; presiunea în pneuri ;
- pentru trenuri : sarcina maximă remorcabilă ; autonomia de mers etc.
- pentru autoturisme : numărul de portiere ; numărul de locuri ;
- pentru camioane : sarcina utilă ; numărul de locuri în cabină ;
- pentru motociclete : tipul de transmisie (lanţ sau cardan).

După cum notează şi autorii, acest gen de ierarhie este unul de tip arbore, iar toţi descendenţii unui obiect de tip „nod" sunt disjuncţi. Modul în care sunt construite relaţiile din ieharhie este însă discutabil. Astfel, autorii propun construirea a câte unei relaţii pentru fiecare nod din ierarhie – un obiect ce are subordonaţi ; astfel, corespunzătoare ierarhiei din figura 8.1, schema propusă ar avea structura :

VEHICULE {IdVehicul, DenVehicul, Producător, Greutate, Preț, MediuNavigare[4]}
VEHICULE_AERIENE {IdVehicul, DenVehicul, Producător, Greutate, Preț, OreAutonomie, H_Max_Zbor[5], CapacitRezervor, VitezaMax, TipCarburant, CategorieVA[6]}
AVIOANE {IdVehicul, DenVehicul, Producător, Greutate, Preț, OreAutonomie, H_Max_Zbor, CapacitRezervor, VitezaMax, TipCarburant, UnghiMaxDecolare, LungPistă}
la care se adaugă VEHICULE_TERESTRE{...}, AUTOVEHICULE {...}, VEHICULE_ACVATICE {...} şi VEHICULE_FLUVIALE {...}.

Relaţia VEHICULE va conţine date despre toate vehiculele din bază, VEHICULE_AERIENE va avea, la un moment dat, un număr de linii egal cu cel al avioanelor din dotare etc. Ceea ce se poate reproşa acestui mod de construire a relaţiilor este gradul mare de redundanţă. Astfel, spre exemplu, denumirea, producătorul, greutatea şi preţul unui avion utilitar apar în trei relaţii. Avantajul ţine de faptul că fiecare relaţie, ce reprezintă o „specializare" a „clasei" VEHICULE, are o structură şi un conţinut care-i conferă o oarecare independenţă faţă de relaţiile „superioare" (cu un grad mai mare de generalitate) şi „inferioare" (specializări ale sale).

Dacă însă am încerca să folosim graful dependenţelor, în care fiecare nivel ierarhic este reprezentat prin dependenţe de incluziune, am obţine o configuraţie precum cea din figura 8.2. Fiecare tip/subtip este identificat prin propriul atribut: IdVehicul ⟶ DenVehicul, IdVehicul ⟶ Producător, ..., IdVehiculAer ⟶ OreAutonomie, IdVehiculAer ⟶ CapacitRezervor... De data aceasta, în dependenţele funcţionale, fiecare identificator determină numai caracteristicile proprii obiectului (specializării) pe care îl reprezintă, iar specializarea este semnalată prin dependenţele de incluziune: IdVehiculAer ⊆ IdVehicul, IdAvioane ⊆ IdVehiculAer, IdAvionPasageri ⊆ IdAvioane etc.

După sistemul deja curent de decupare a relaţiilor, izolând fiecare sursă de dependenţe, am obţine relaţiile:

VEHICULE {IdVehicul, DenVehicul, Producător, Greutate, Preț}
VEHICULE_AERIENE {IdVehicAer, OreAutonomie, H_Max_Zbor, CapacitRezervor, VitezaMax, TipCarburant}
AVIOANE {IdAvion, UnghiMaxDecolare, LungPistă}
AVIOANE_PASAGERI {IdAvionPasageri, NrPasageri}
AVIOANE_UTILITARE {IdAvionUtilitar, SarcinăUtilă}
ELICOPTERE {IdElicopter, NrPaleteEliceP, TipRotor}
VEHICULE_TERESTRE {IdVehicTerestru, VitezăMax, TipCarburant}

4. MediuNavigare poate avea una dintre valorile: *terestru*, *aer*, *apă*.
5. Înălţimea maximă de zbor.
6. Domeniul atributului CategorieVA este alcătuit din valorile *avion* şi *elicopter*.

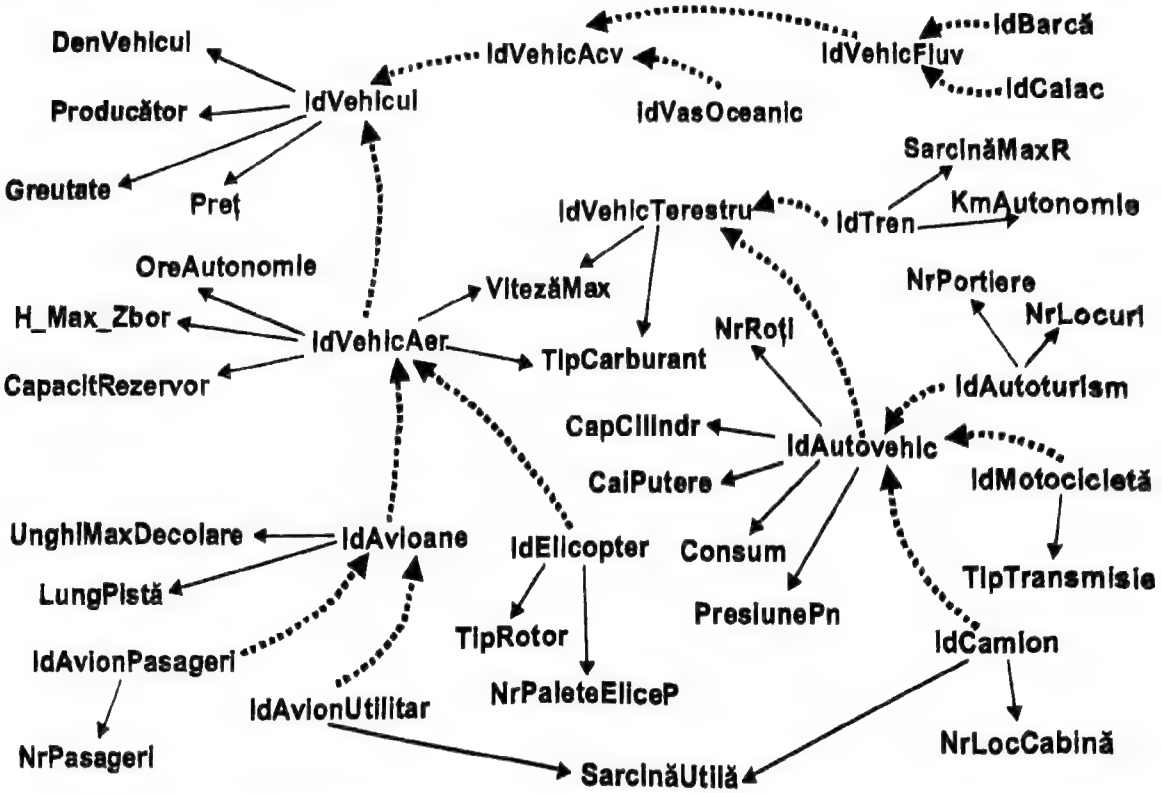


Figura 8.2. Reprezentarea generalizării/specializării prin dependențe de incluziune

TRENURI {IdTren, SarcinăMaxR, KmAutonomie}
AUTOVEHICULE {IdAutovehic, NrRoți, CapCilindr, CaiPutere, Consum, PresiunePn}
MOTOCICLETE {IdMotocicletă, TipTransmisie}
AUTOTURISME {IdAutoturism, NrPortiere, NrLocuri}
CAMIOANE {IdCamion, NrLocCabină, SarcinăUtilă}
VEHICULE_ACVATICE {IdVehicAcv}
VASE_OCEANICE {IdVasOceanic}
VEHICULE_FLUVIALE {IdVehicFluv}
BĂRCI {IdBarcă}
CAIACE {IdCaiac}

Chiar dacă nu este implicată ca sursă în nici o dependenţă funcţională, pentru orice sursă de dependenţă de incluziune se constituie câte o relaţie. Aşa „răsar" relaţiile VEHICULE_ACVATICE, VASE_OCEANICE, VEHICULE_FLUVIALE, BĂRCI, CAIACE. Restricţiile referenţiale se stabilesc între:

- VEHICULE_AERIENE.IdVehiculAer (copil) şi VEHICULE.idVehicul (părinte);
- VEHICULE_TERESTRE.IdVehicTerestru (copil) şi VEHICULE.IdVehicul (părinte);
- VEHICULE_ACVATICE.IdVehicAcv (copil) şi VEHICULE.IdVehicul (părinte);
- AVIOANE.IdAvion (copil) şi VEHICULE_AERIENE.IdVehiculAer (părinte);

- AVIOANE_PASAGERI.IdAvionPasageri (copil) și AVIOANE.IdAvion (părinte);
- AVIOANE_UTILITARE.IdAvionUtilitar (copil) și AVIOANE.IdAvion (părinte);
- ELICOPTERE.IdElicopter (copil) și VEHICULE_AERIENE.IdVehiculAer (părinte);
- TRENURI.IdTren (copil) și VEHICULE_TERESTRE.IdVehicTerestru (părinte);
- AUTOVEHICULE.IdAutovehic (copil) și VEHICULE_TERESTRE.IdVehicTerestru (părinte);
- AUTOTURISME.IdAutoturism (copil) și AUTOVEHICULE.IdAutovehic (părinte);
- MOTOCICLETE.IdMotocicletă (copil) și AUTOVEHICULE.IdAutovehic (părinte);
- CAMIOANE.IdCamion (copil) și AUTOVEHICULE.IdAutovehic (părinte);
- VASE_OCEANICE.IdVasOceanic (copil) și VEHICULE_ACVATICE.IdVehicAcv (părinte);
- VASE_FLUVIALE.IdVasFluvial (copil) și VEHICULE_ACVATICE.IdVehicAcv (părinte);
- BĂRCI.IdBarcă (copil) și VASE_FLUVIALE.IdVasFluvial (părinte);
- CAIACE.IdCaiac (copil) și VASE_FLUVIALE.IdVasFluvial (părinte).

Figura 8.3 este mult mai sugestivă, pentru un plus de inteligibilitate unindu-se cheile-copil cu cele părinte.

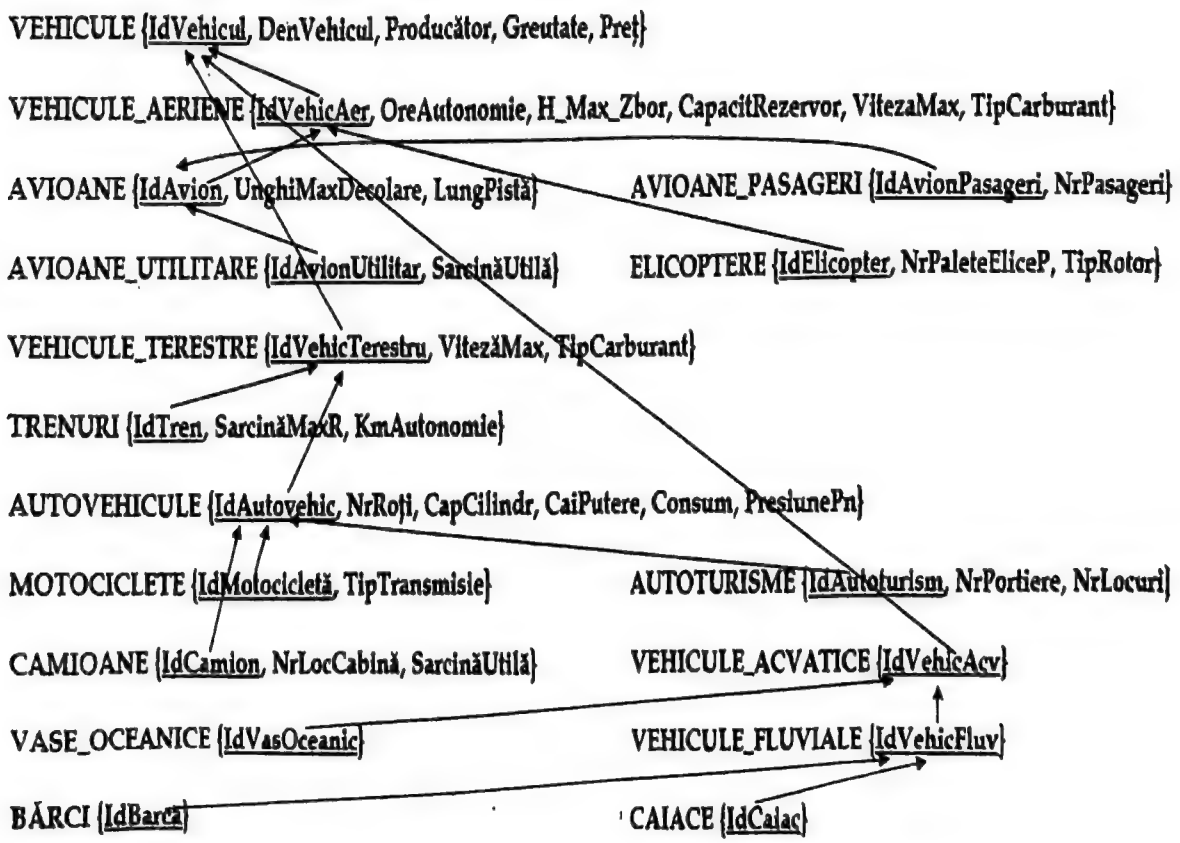


Figura 8.3. Semnalizarea restricțiilor referențiale

Schema bazei reflectă cu acurateţe specializarea tipurilor de vehicule. Ceea ce i se poate reproşa este fragmentarea excesivă. Putem îndulci această senzaţie dacă, pentru toţi „părinţii" care au „descendenţi" fără atribute proprii, introducem un atribut de genul `TipCopil`. Astfel, în loc de triada VEHICULE_FLUVIALE {IdVehicFluv} - BĂRCI {IdBarcă} - CAIACE {IdCaiac}, am folosi doar prima relaţie, cu schema VEHICULE_FLUVIALE {IdVehicFluv, TipVehicFluv}, unde `TipVehicFluv` poate lua doar valorile *Barcă* şi *Caiac*. Economia este mai degrabă nesemnificativă în cazul nostru, deşi ceva mai importantă în alte situaţii.

Un alt reproş adresabil schemei ţine de relativa dificultate în obţinerea unor informaţii. Unele interogări sunt relativ simple, altele necesită destul de multe joncţiuni. De exemplu, dacă dorim să obţinem lista producătorilor de bărci, interogarea folosită ar fi cât se poate de simplă:

```
SELECT DISTINCT Producător
FROM bărci b
   INNER JOIN vehicule v ON b.idbarcă = v.idvehicul
```

Dacă însă interesează *ce categorii de vehicule produce firma X*, lucrurile se complică întrucâtva, nu atât ca sintaxă şi dificultate, cât din punctul de vedere al lungimii interogării:

```
SELECT DISTINCT 'Avioane pasageri' AS Categorie
FROM vehicule v INNER JOIN avioane_pasageri ap
   ON v.idvehicul = ap.idavionpasageri
WHERE producător='X'
UNION
SELECT DISTINCT 'Avioane utilitare' AS Categorie
FROM vehicule v INNER JOIN avioane_utilitare au
   ON v.idvehicul = au.idavionutilitar
WHERE producător='X'
UNION
SELECT DISTINCT 'Elicoptere' AS Categorie
FROM vehicule v INNER JOIN elicoptere e
   ON v.idvehicul = e.idelicopter
WHERE producător='X'
UNION
SELECT DISTINCT 'Autoturisme' AS Categorie
FROM vehicule v INNER JOIN autoturisme a
   ON v.idvehicul = a.idautoturism
WHERE producător='X'
UNION
SELECT DISTINCT 'Camioane' AS Categorie
FROM vehicule v INNER JOIN camioane c
   ON v.idvehicul = c.idcamion
```

```
WHERE producător='X'
UNION
SELECT DISTINCT 'Motociclete' AS Categorie
FROM vehicule v INNER JOIN motociclete m
  ON v.idvehicul = m.idmotocicletă
WHERE producător='X'
UNION
SELECT DISTINCT 'Trenuri' AS Categorie
FROM vehicule v INNER JOIN trenuri t
  ON v.idvehicul = t.idtren
WHERE producător='X'
UNION
SELECT DISTINCT 'Vase oceanice' AS Categorie
FROM vehicule v INNER JOIN vase_oceanice vo
  ON v.idvehicul = vo.idvasoceanic
WHERE producător='X'
UNION
SELECT DISTINCT 'Caiace' AS Categorie
FROM vehicule v INNER JOIN caiace c
  ON v.idvehicul = c.idcaiac
WHERE producător='X'
UNION
SELECT DISTINCT 'Bărci' AS Categorie
FROM vehicule v INNER JOIN bărci b
  ON v.idvehicul = b.idbarcă
WHERE producător='X'
```

Specializare multiplă

Specific ierarhiei vehiculelor din figura 8.2 este de faptul că toată structura sa este perfect arborescentă, specializarea fiecărui nivel inferior unui nod fiind întotdeauna una simplă. De multe ori însă un nod în ierarhie poate împrumuta (sau moşteni) proprietăţi nu numai ale unui singur părinte, ci şi ale două sau mai multe noduri. Să examinăm ierarhia din figura 8.4 ce reprezintă structura personalului la o universitate de tipul Universităţii „Al.I. Cuza" din Iaşi.

Activităţile din cadrul universităţii, în principal activităţile didactice pot fi derulate atât de personalul *propriu* al universităţii, cât şi de *colaboratori* externi, cei care au carte de muncă în alte instituţii sau firme (de exemplu: redactori la reviste de cultură pot preda cursuri la Facultatea de Litere). Cursurile pot fi predate doar de *profesori*, *conferenţiari* şi *lectori*, care sunt grupaţi în „nodul" *curs_cd* (de la *titulari de cursuri dintre cadrele didactice*), la care se adaugă o parte dintre colaboratori, mai precis „partea" *curs_colab* (de la *colaboratori ce pot ţine cursuri*). Seminariile pot fi ţinute de toate cadrele didactice şi de doctoranzii afiliaţi catedrelor, dar şi de o subcategorie de analişti/programatori (`an/prog`), numită `an/pr_sem` (de la *analişti/programatori ce pot ţine seminarii*), plus o parte dintre colaboratori – `sem_colab`.

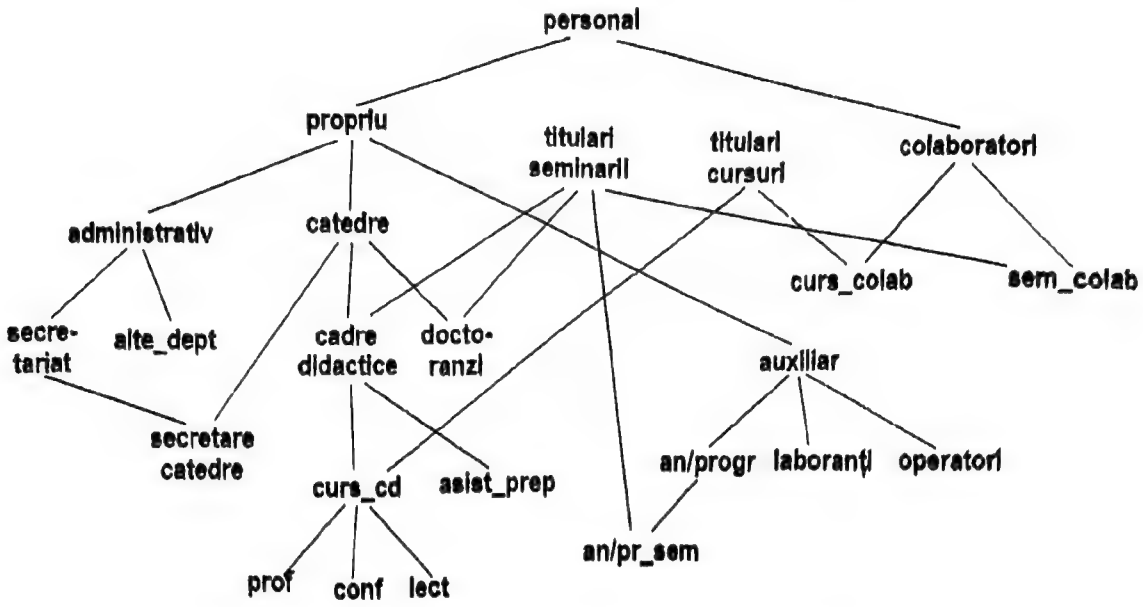


Figura 8.4. Structura personalului unei universități

La fiecare catedră există o secretară. Aceasta moștenește atât atributele categoriei profesionale din care face parte, SECRETARE: `VitezăTehn` (viteză la tehnoredactarea documentelor, adică număr de cuvinte pe minut), `LimbăStrăină` (fiecare limbă străină pe care o cunoaște), `NivelLS` (nivel de competență pentru limba respectivă), cât și atributele catedrei de care aparține: `DenCatedră`, `SediuCatedră`, `TelCatedră`.

Datele generale despre fiecare angajat/colaborator sunt: `CNP` (codul numeric personal), `NumePren`, `Adresa`, `Telefon`, `EMail`. Pentru toți colaboratorii interesează locul de muncă (`LocDeMuncă`) și profesia (`Profesie`), în timp ce tuturor angajaților proprii li se stochează marca (`Marcă`), data angajării în universitate (`DataAngajării`) și data nașterii (`DataNașterii`).

Despre angajații din categoria personalului administrativ (`MarcăAdmin`), aici încadrându-se atât secretarele, cât și cei de la alte departamente, interesează postul pe care sunt angajați (`Post`) și compartimentul universității sau facultatea de care aparțin (`Compart_Fac`). Secretarele sunt o specializare a personalului administrativ cu cele trei atribute menționate mai sus.

Informațiile specifice personalului auxiliar (`MarcăAux`) sunt: calificarea (`Calificare`) și categoria de salarizare (`Categorie`), însă datorită faptului că laboranții și operatorii nu au alte atribute specifice, celor două atribute li se mai adaugă unul tocmai pentru stabilirea tipului de personal auxiliar din care face parte un asemenea angajat (`TipAux`). Pentru „subspecia" analiști/programatori (`MarcăAP`) este important de știut în ce platforme software (`Software`) are competențe fiecare angajat, mai precis nivelul (`Nivel`) și numărul de ani de experiență (`AniExperiență`).

Personalul didactic este structurat pe catedre (`IdCatedră`), pentru care interesează: denumirea (`DenCatedră`), camera/biroul ce reprezintă sediul catedrei (`SediuCatedră`) și numărul de telefon al catedrei (`TelCatedră`). Tot în categoria membrilor de catedră intră și doctoranzii „cu frecvență" (`MarcăDrd`), care nu sunt cadre

didactice propriu-zise, dar pot ține seminarii. Domeniul de doctorat (`DomeniuDoctorat`) și anul absolvirii facultății de către doctorand (`AnAbsolvFacult`) sunt două atribute specifice numai „subclasei" *doctoranzi*.

Pentru toate cadrele didactice dintr-o catedră (`MarcăDidact`) interesează gradul didactic (`GradDidactic`), care poate fi *preparator*, *asistent*, *lector*, *conferențiar* sau *profesor*, și data ultimei promovări (`DataUltimeiPromovări`). Astfel, sunt preluați în bază toți membrii catedrei, iar calitatea automată a tuturor cadrelor didactice de a ține seminarii este reprezentată prin dependența funcțională `MarcăDidact` ⟶ `IdTitularSem`. Deoarece numai o parte dintre cadrele didactice pot ține cursuri, se poate recurge la o subclasă a cadrelor didactice, numită *curs_cd*, de aici și identificatorul `MarcăDidCurs` (marca pentru cadrul didactic ce poate ține cursuri).

Datele specifice fiecărui titular de curs (`IdTitularCurs`) sunt numărul de discipline la care ține curs (`NrCursuri`) și numărul de serii de curs (`NrSeriiCurs`), în timp ce numărul de grupe de seminar (`NrGrupeSem`) reprezintă singura informație specifică titularilor de seminarii (`IdTitularSem`).

Graful tuturor dependențelor funcționale și de incluziune este cel din figura 8.5.

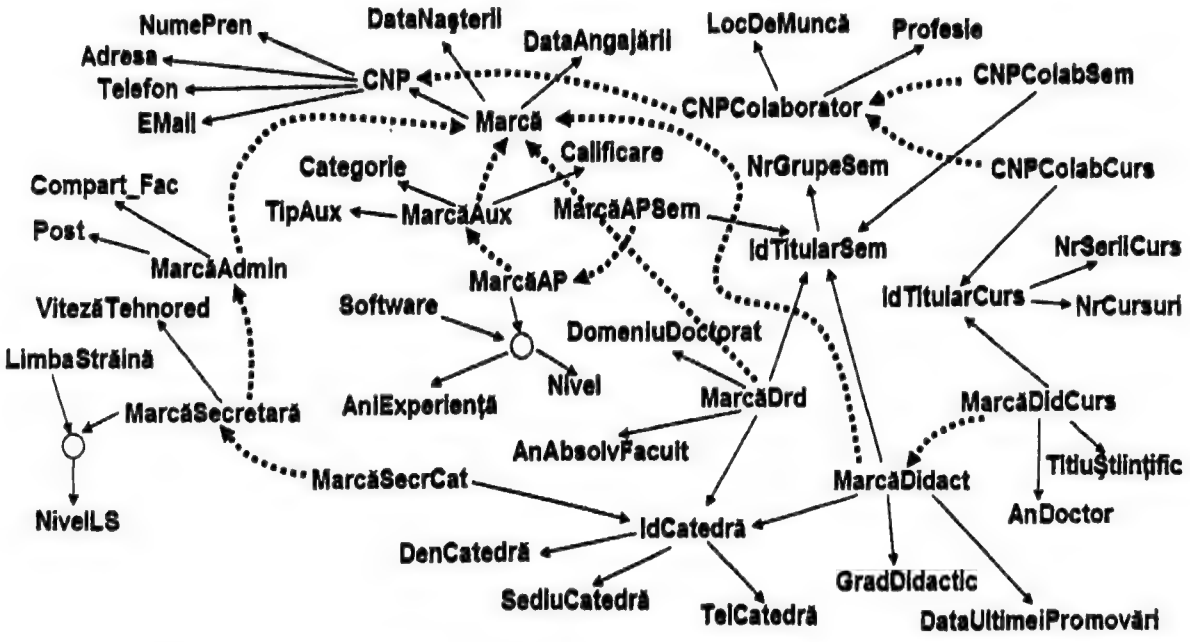


Figura 8.5. Graful dependențelor pentru baza de date PERSONAL_UNIVERSITATE

Ceea ce ne interesează în cel mai înalt grad este reprezentarea specializărilor, unele simple, altele multiple, ale diferitelor categorii de personal. Specializările simple sunt reprezentate, în general, prin dependențe de incluziune:

- `CNPColaborator` ⊆ `CNP` înseamnă că valorile codurilor numerice personale pentru colaboratorii externi reprezintă un subset al valorilor codurilor numerice personale pentru toate persoanele avute în evidența universității;
- `MarcăAdmin` ⊆ `Marcă`, `MarcăAux` ⊆ `Marcă`, `MarcăDidact` ⊆ `Marcă` și `MarcăDrd` ⊆ `Marcă` ilustrează categoriile principale ale personalului propriu;
- `CNPColabCurs` ⊆ `CNPColaborator` și `CNPColabSem` ⊆ `CNPColaborator` sugerează specializările categoriei *colaboratori*;

- MarcăSecretară ⊆ MarcăAdmin indică includerea secretarelor în rândul personalului administrativ;
- MarcăSecrCat ⊆ MarcăSecretară: valorile mărcilor tuturor secretarelor de catedră se regăsesc în valorile mărcilor secretarelor;
- MarcăAP ⊆ MarcăAux: analiştii/programatorii sunt personal auxiliar;
- MarcăAPSem ⊆ MarcăAP: numai o parte dintre analişti/programatori pot ţine seminarii;
- MarcăDidCurs ⊆ MarcăDidact: numai o parte dintre cadrele didactice dintr-o catedră pot ţine cursuri.

Faptul că un angajat propriu, identificat prin Marcă, se regăseşte în clasa-rădăcină *personal* este indicat prin dependenţa funcţională Marcă ⟶ CNP.

Între specializările multiple din graf putem identifica două tipologii. Pe de o parte, faptul că o secretară este arondată unei catedre şi, totodată, face parte din categoria *secretare* se rezolvă printr-o dependenţă funcţională, MarcăSecrCat ⟶ IdCatedră, coroborată cu una de incluziune, MarcăSecrCat ⊆ MarcăSecretară. Pe de altă parte, problema fundamentală a ierarhiei a fost că două dintre clase, *titulari_cursuri* şi *titulari_seminarii*, au o structură cu totul eterogenă. Astfel titularii de cursuri sunt o parte dintre cadrele didactice, identificate prin marcă, plus o parte dintre colaboratori, la care identificatorul este CNP-ul. Specializarea multiplă s-a realizat prin combinarea a două *dependenţe de incluziune*, MarcăDidCurs ⊆ MarcăDidact şi CNPColabCurs ⊆ CNPColaborator, cu două *dependenţe funcţionale*: CNPColabCurs ⟶ IdTitularCurs şi MarcaDidCurs ⟶ IdTitularCurs. În ambele tipuri de probleme soluţia a constituit-o combinaţia dependenţe de incluziune-dependenţe funcţionale.

Din graf, izolând câte o relaţie pentru fiecare sursă de DF/DI, obţinem structura din figura 8.6. Singura tabelă care nu corespunde unei surse de DF este AN_PROG (analişti-programatori), care este însă necesară pentru a reflecta această categorie în cadrul personalului auxiliar, deci pentru a îngloba în schemă dependenţa de incluziune MarcăAP ⊆ MarcăAux.

Structura pare a fi una rezonabilă, chiar dacă numărul relaţiilor este destul de mare. Este drept, unele operaţiuni se traduc prin actualizarea simultană a mai multor tabele. Spre exemplu, un asistent universitar devine, în urma unui concurs, lector (la fel de universitar). Noua categorie didactică îi permite să fie titular de curs (nici nu ar putea fi declarat lector dacă nu ar avea în norma didactică un număr minim de ore de curs). Prin urmare, schimbarea gradului didactic din asistent în lector presupune:

- adăugarea unei linii în tabela TITULARI_CURSURI;
- modificarea valorii atributului CADRE_DIDACTICE.GradDidactic;
- inserarea unei linii în tabela CD_TITUL_CURS.

Deşi complicat, acest mod de lucru poate fi gestionat riguros, iar informaţiile furnizate de baza de date sunt conforme cu realitatea.

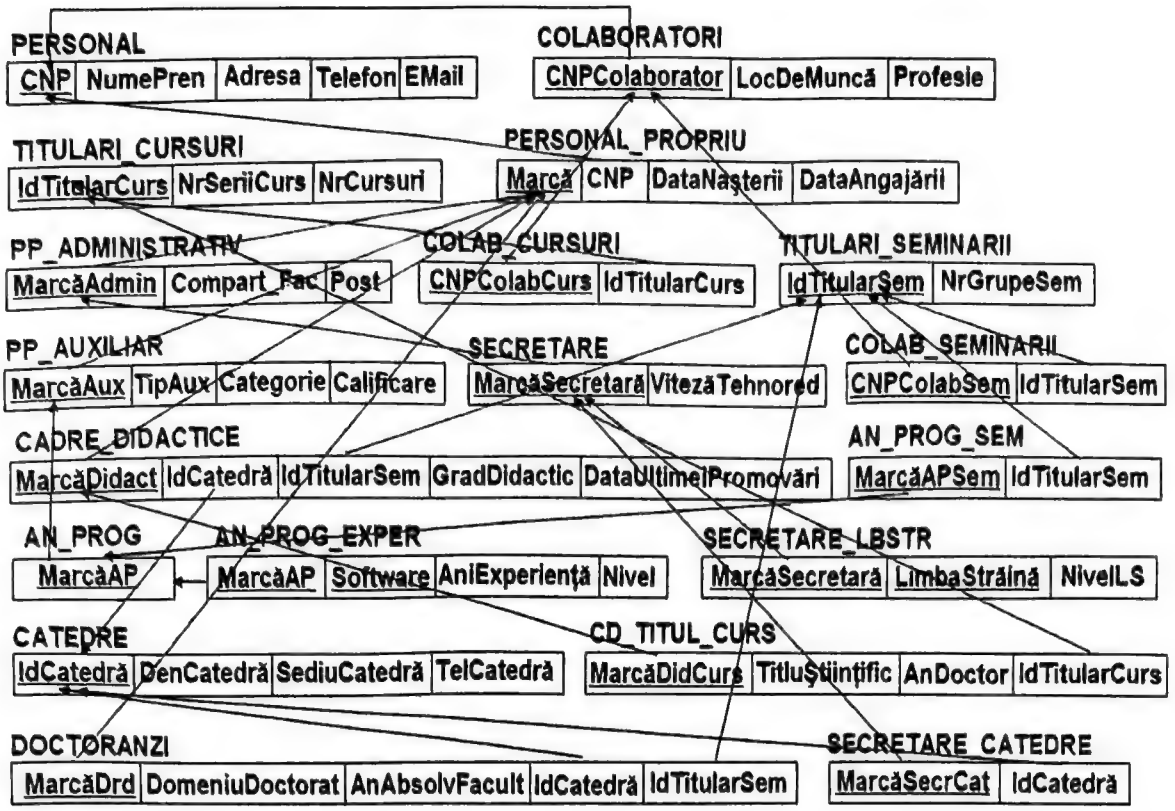


Figura 8.6. Schema bazei de date PERSONAL_UNIVERSITATE

8.2. Ierarhii generalizate

Problema generalizării/specializării, aşa cum a fost prezentată în paragraful anterior, prezintă importanţă pentru schema bazei de date atunci când:

- numărul palierelor structurii ierarhice nu este prea mare;
- atât nodurile de pe acelaşi nivel ierarhic, cât şi nodurile subordonate au o serie de caracteristici proprii, inaplicabile altor categorii de noduri;
- ierarhia prezintă o considerabilă constanţă în timp.

Cu totul altfel stau lucrurile la structurile ierahice dinamice, în care specializarea poate să se facă pe un număr de niveluri impresionant (de mare). În aceste cazuri trebuie găsite soluţii care să asigure gradul de flexibilitate necesar. Să luăm un caz real. Firmele cu un mare număr de angajaţi şi o structură a posturilor destul de încărcată se confruntă cu una dintre problemele managementului resurselor umane, şi anume gestiunea competenţelor. În funcţie de natura activităţii, firmele pot dezvolta un nomenclator al competenţelor, organizat ierarhic ca în figura 8.7.

Pentru a păstra lizibilă figura, majoritatea competenţelor nu au mai fost dezvoltate pe subcategorii. Principalele clase de competenţe sunt:

1. *Tehnico-profesionale*, cu subclasele: *proiectare*, *electrician auto*, *financiar*, *marketing*, *contabilitate* etc. Fiecare poate fi descompusă pe oricâte niveluri, aşa cum, spre exemplu, competenţele contabile se defalchează în *contabilitate*

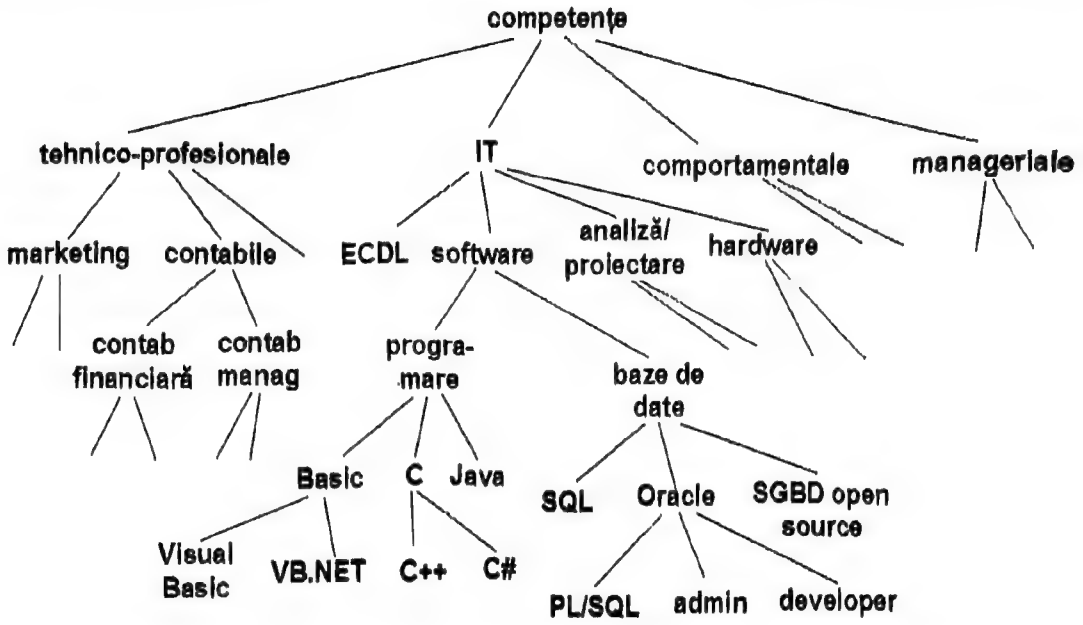


Figura 8.7. O porțiune din ierarhia competențelor într-o organizație

financiară și *contabilitate managerială*. Competențele ce țin de contabilitatea financiară se pot descompune în categoriile: *imobilizări*, *stocuri*, *trezorerie* etc.

2. *IT* – cele mai atent tratate în figură, deși nu sută la sută riguros. Pentru majoritatea posturilor non-informatice, nivelul ECDL (*European Computer Driving Licence* – permisul european de „conducere" a calculatorului) este unul suficient, deoarece presupune noțiuni elementare/medii despre: arhitectura calculatoarelor, sisteme de operare (Windows), procesare de text (Word), foi de calcul (Excel), prezentări (PowerPoint), baze de date (Access) și Internet. Posturile informatice reclamă însă o mult mai minuțioasă descompunere, așa încât au mai fost adăugate trei categorii: hardware, software și analiză/ proiectare. La software interesează cunoștințele de programare și baze de date, fiecare categorie fiind, la rând său, descompusă.
3. *Comportamentale* – privesc, printre altele: răbdarea, tenacitatea, abilitățile pentru lucrul în echipă, comunicarea etc.
4. *Manageriale* – se referă la abilități de antrenare și motivare, rezolvarea conflictelor, organizarea echipei, managementul proiectelor etc.

Reamintim că schema de mai sus nu este una ideală (nici nu cred că v-ați făcut iluzii), ci reflectă interesele unei organizații la un moment dat. Nu este exclus, spre exemplu, ca la anumite competențe să apară subcategorii (de exemplu: competențele *IT* → *software* → *baze de date* → *Oracle* → *developer* să fie descompuse în categoria *Forms* și o alta *Reports*; sau competențele *IT* → *software* → *baze de date* → *SGBD open source* să se decompună în *MySQL*, *PostgreSQL*, *Firebird* etc.).

Avantajul structurii ierarhice din figură ține de faptul că fiecare competență, indiferent de tipul și nivelul pe care se află, se caracterizează prin: denumire (`DenCompetență`), unitate de măsură (`UM`) și modul în care se face evaluarea sa (`ModEvaluare`). La aceste atribute, mai trebuie adăugate două, unul pentru identificarea

competenței (`IdCompetență`) și un altul ce semnalează competența superioară (`IdCompSuperioară`). Pe baza acestor atribute se poate cunoaște în orice moment întreaga arborescență a competențelor. Ca și la discuțiile din capitolul precedent privitoare la conturile contabile, o parte dintre competențe sunt elementare („frunză"), în sensul că nu se descompun, în timp ce altele sunt „ramuri" (rîuri...).

În afara plăcerii discuției în sine despre competențe, există și un motiv plauzibil pentru care am zăbovit asupra acestui subiect: firmele au o organigramă pe care sunt trecute diverse categorii de posturi, iar fiecare post (*ajutor analist-proiectant 1*, *dezvoltator de aplicații - 1*, *dezvoltator de aplicații - 2*, *administrator de baze de date* etc.), identificat prin atributul `IdPost`, având o denumire (`DenPost`) și un nivel de salarizare (`Salariu`) stabilite, necesită anumite categorii de competențe, fiecare competență la un nivel minim acceptabil ce depinde chiar de natura postului.

Exemple:

- postul *dezvoltator de aplicații - 1* necesită următoarele competențe:
  - VB.Net - nivel minim acceptat 9 (din 10);
  - SQL - nivel minim acceptat 8;
  - Oracle PL/SQL - nivel minim acceptat 8;
  - abilități de lucru în echipă - nivel minim acceptat 7;
- postul *dezvoltator de aplicații - 2* necesită următoarele competențe:
  - SQL - nivel minim acceptat 7;
  - Oracle PL/SQL - nivel minim acceptat 9;
  - Java - nivel minim acceptat 9;
  - abilități de lucru în echipă - nivel minim acceptat 8.

Ca restricție importantă a bazei, stabilim că descrierea posturilor se realizează exclusiv prin competențe „frunză", adică elementare, altminteri ar fi destul de greu de stabilit ce ar reprezenta nivelul minim acceptat.

Prin combinarea chestiunilor legate de competențe și posturi, cu câteva elemente sumare ce țin de angajați (marcă, nume, data angajării, compartimentul în care lucrează), compartimente (șeful acestuia) și evaluarea competențelor angajaților (nivelul), se poate obține un graf al dependențelor precum cel din figura 8.8.

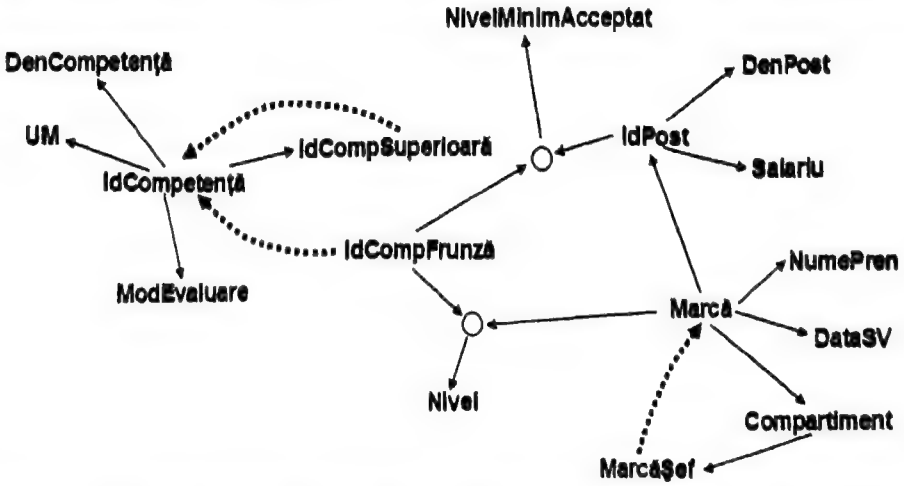


Figura 8.8. Graful DF/DI pentru gestiunea competențelor pe posturi și angajați

Există trei dependenţe de incluziune: prima, pe care am întâlnit-o şi în capitolul anterior, semnifică faptul că valorile `MarcăŞef` sunt un subset al valorilor atributului `Marcă`. La fel şi `IdCompSuperioară` *versus* `IdCompetenţă` şi `IdCompFrunză` *versus* `IdCompetenţă`. Am recurs la „specializarea" `IdCompFrunză` tocmai pentru a gestiona mai bine restricţia după care în evaluarea unui post sau angajat pot fi implicate numai competenţele elementare.

Nu ne mai rămâne decât să decupăm relaţiile şi să le „botezăm", schema fiind cea din figura 8.9 Scriptul Oracle de creare a tabelelor poate fi descărat de pe Web (*http : // www.feaa.uaic.ro/cercetare/publicatii*) sub forma *listing*-ului 8.1. Deşi nu este cea mai simplă soluţie, ne încăpăţânăm să folosim o tabelă specială pentru competenţele ce pot fi evaluate şi incluse în descrierea unui post (COMPETENŢE_ELEMENTARE). Consideraţiile sunt similare celor din paragraful 7.4 (vezi figura 7.9).

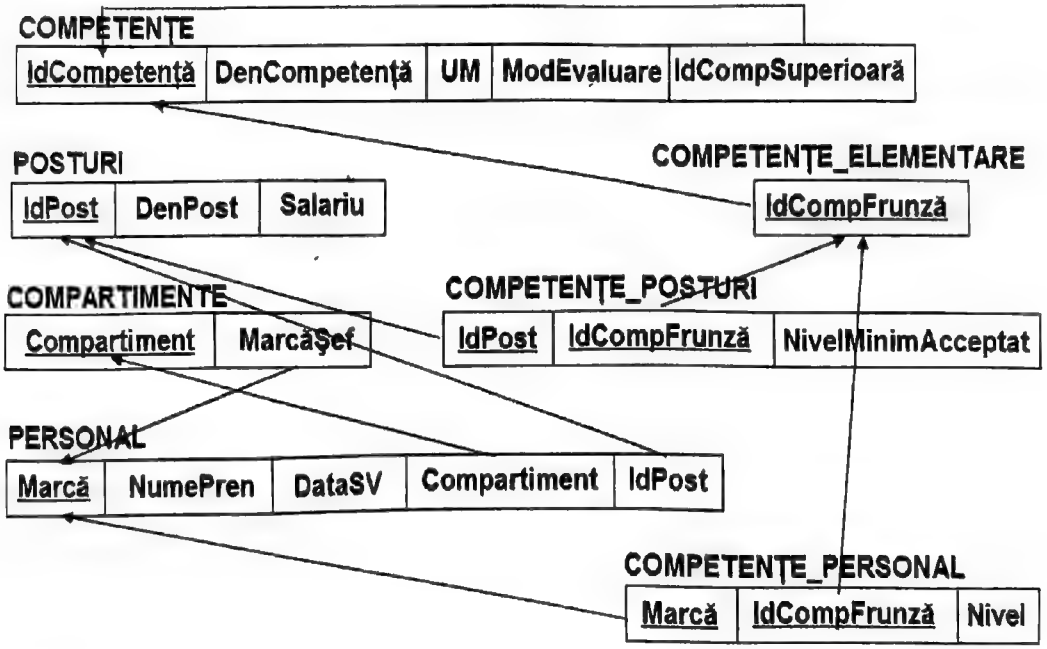


Figura 8.9. Schema obţinută pe baza grafului din figura 8.8

Spre deosebire de planul de conturi, ierarhia competenţelor reiese nu din identificator, ci din folosirea unui atribut special dedicat – `IdCompSuperioară`. Ceea ce ne interesează e faptul că, neavând un regim atât de draconic precum înregistrările contabile, am putea folosi o serie de artificii „liber alese".

Să luăm un caz la întâmplare. În ierarhia din figura 8.7, competenţele legate de SQL sunt elementare (frunză). Dar, după ceva timp, firma realizează că în unele proiecte are nevoie de SQL-işti care să lucreze cu obiecte, funcţii OLAP şi alte opţiuni care sunt prezente începând cu standardul SQL:1999. Soluţia necesită descompunerea competenţelor *SQL* în două subcategorii: *SQL-92* şi *SQL:1999*. Bun, dar ce facem cu posturile care au deja descrise, de ani buni, în cerinţele lor competenţe *SQL*? Plus angajaţii care au fost evaluaţi până acum la categoria „frunză" *SQL*? Ei bine, o idee nu foarte respingătoare este ca, o dată cu introducerea primei subcategorii SQL, să inserăm automat categoria *SQL_Veche*, prin care să sugerăm că este o competenţă elementară artificială, necesară păstrării datelor deja existente

în tabelele COMPETENŢE_POSTURI şi COMPETENŢE_PERSONAL. Noua ramificaţie ar fi cea din figura 8.10.

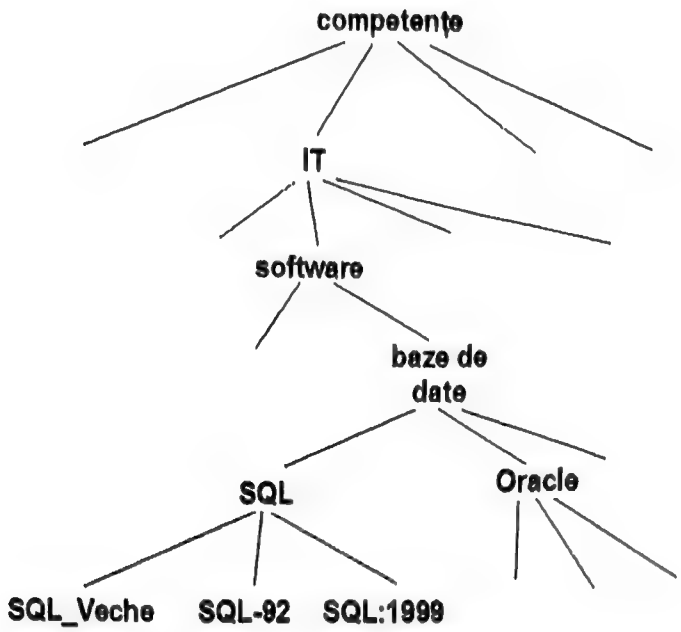


Figura 8.10. Descompunerea „competenţei" SQL

Atributul `IdCompetenţă` a fost gândit ca o cheie surogat, deci scenariul adăugării primei competenţe dintre *SQL-92* sau *SQL : 1999* ar fi următorul:

- se inserează o înregistrare în COMPETENŢE corespunzătoare *SQL_Veche*, înregistrare care va avea o valoare a `IdCompetenţă` pe care o vom nota `IdCompetenţă(SQL_Veche)` şi, de asemenea:
  - `DenCompetenţă(SQL_Veche) := DenCompetenţă(SQL) || '_Veche';`
  - `UM(SQL_Veche) := UM(SQL);`
  - `ModEvaluare(SQL_Veche) := ModEvaluare(SQL);`
  - `IdCompSuperioară(SQL_Veche) := ICompetenţă(SQL);`
- în tabela COMPETENŢE_ELEMENTARE, valoarea `IdCompFrunză(SQL)` se înlocuieşte cu `IdCompetenţă(SQL_Veche)`. Datorită restricţiilor referenţiale (sau declanşatorului de modificare a tabelei) se vor modifica în „cascadă", în tabelele COMPETENŢE_POSTURI şi COMPETENŢE_PERSONAL, toate valorile `IdCompetenţă(SQL)` cu `IdCompetenţă(SQL_Veche)`.

Implementarea acestui mecanism reclamă folosirea unor funcţii pe care le vom crea după modelul Oracle PL/SQL, ca şi în alte ocazii. Puţin ciudat, poate, nu ne vom apuca de treabă până nu vom fi implementat declanşatoarele de actualizare în cascadă a cheilor primare din tabelele-părinte (vezi *listing*-ul 8.2, ce poate fi descărcat de pe Web). Continuăm cu discutarea primei funcţii, f_este_compfrunză – vezi *listing*-ul 8.3 –, care primeşte ca argument un identificator de competenţă şi returnează TRUE în cazul în care competenţa respectivă este una elementară şi FALSE în caz contrar.

Listing 8.3. Funcţia PL/SQL f_este_compfrunză

```
CREATE OR REPLACE FUNCTION f_este_compfrunza (
id_competenta NUMBER) RETURN BOOLEAN
AS
v_unu NUMBER(1) ;
BEGIN
SELECT 1 INTO v_unu FROM dual WHERE EXISTS
    (SELECT 1 FROM competente_elementare
     WHERE IdCompFrunza = id_competenta );
RETURN TRUE ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RETURN FALSE ;
END ;
```

A doua funcţie verifică dacă o competenţă a fost deja folosită (măcar o dată) la descrierea competenţelor unui post sau ale unui angajat – f_deja_folosită –, vezi *listing*-ul 8.4.

Listing 8.4. Funcţie ce verifică folosirea unei competenţe

```
CREATE OR REPLACE FUNCTION f_deja_folosita (
 id_competenta NUMBER) RETURN BOOLEAN
AS
v_unu NUMBER(1) ;
BEGIN
SELECT 1 INTO v_unu FROM dual WHERE EXISTS
    (SELECT 1 FROM competente_posturi   WHERE IdCompFrunza = id_competenta
  UNION
  SELECT 1 FROM competente_personal WHERE IdCompFrunza = id_competenta
   );
RETURN TRUE ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RETURN FALSE ;
END ;
```

Cele două funcţii ne vor fi de folos în declanşatoarele de inserare în COMPETENŢE (sunt două, unul de tip *before* şi altul *after*), în care se va verifica dacă „părintele" competenţei adăugate este, în momentul inserării, competenţă elementară. În caz că da, se obţine identificatorul competenţei „artificiu", care se adaugă în tabelă preluând valorile atributelor din cele ale „părintelui", apoi se modifică identificatorul în COMPETENŢE_ELEMENTARE.

Listing 8.5. Un pachet şi două declanşatoare pentru inserare în COMPETENŢE

```
CREATE OR REPLACE PACKAGE pac_competente
AS
 v_intra BOOLEAN := TRUE ;
```

```
 v_competente competente%ROWTYPE ;
END pac_competente ;
/

-------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_competente_ins_before
 BEFORE INSERT ON competente FOR EACH ROW
BEGIN
   IF pac_competente.v_intra = TRUE THEN
     SELECT seq_idcompetente.NextVal INTO :NEW.idCompetenta FROM dual ;
     pac_competente.v_competente.idCompetenta := :NEW.idCompetenta ;
     pac_competente.v_competente.idCompSuperioara := :NEW.idCompSuperioara ;
   END IF ;
END ;
/

-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_competente_ins_after
 AFTER INSERT ON competente
DECLARE
  v_id competente.idCompetenta%TYPE ;
BEGIN
   IF pac_competente.v_intra = TRUE THEN
     IF f_este_compfrunza (pac_competente.v_competente.idCompSuperioara) THEN
            -- parintele era "frunza" înaintea inserarii
           IF f_deja_folosita (pac_competente.v_competente.idCompSuperioara) =
           FALSE THEN
               -- competenta parinte nu apare in evaluarea niciunui post/angajat
            UPDATE competente_elementare
               SET idCompFrunza = pac_competente.v_competente.idCompetenta
               WHERE idCompFrunza = pac_competente.v_competente.idCompSuperioara;
         ELSE
       -- competenta parinte este deja folosita in COMPETENTE_POSTURI
          --   sau COMPETENTE_ANGAJATI
            INSERT INTO competente_elementare
               VALUES (pac_competente.v_competente.idCompetenta) ;

            SELECT * INTO pac_competente.v_competente FROM competente
           WHERE idCompetenta =
           pac_competente.v_competente.idCompSuperioara ;

           SELECT seq_idcompetente.NextVal INTO v_id FROM dual ;

           pac_competente.v_intra := FALSE ;
           INSERT INTO competente VALUES (
                v_id, pac_competente.v_competente.DenCompetenta || '_VECHE_',
                pac_competente.v_competente.UM, pac_competente.v_competente.ModEvaluare,
               pac_competente.v_competente.idCompetenta) ;
              pac_competente.v_intra := FALSE ;

           UPDATE competente_elementare SET idCompFrunza = v_id
             WHERE idCompFrunza = pac_competente.v_competente.idCompetenta ;
```

```
        END IF ;
    ELSE
    -- parintele NU era "frunza" inaintea inserarii
    INSERT INTO competente_elementare
          VALUES (pac_competente.v_competente.IdCompetenta) ;
    END IF ;
  END IF ;
END ;
/
```

Soluţia nu poate funcţiona dacă nu este implementat mecanismul de actualizare în cascadă a modificării `COMPETENŢE_ELEMENTARE.IdCompFrunză` în tabelele COMPETENŢE_POSTURI şi COMPETENŢE_PERSONAL (prin comanda `CREATE TABLE... ON UPDATE CASCADE`, acolo unde este posibil, sau prin declanşatoare). Pentru verificarea funcţionării declanşatoarelor vă invit să lansaţi scriptul de populare a tabelelor din bază, ce poate fi descărcat de pe Web de la adresa cunoscută, sub forma *listing*-ului 8.6.

În continuare ne propunem să obţinem câteva informaţii interesante din tabelele bazei. Să începem cu răspunsul la întrebarea: ce competenţe necesită postul *Dezvoltator aplicaţii – 2*? Soluţia este relativ simplă:

```
SELECT DenCompetenta, UM, ModEvaluare, NivelMinimAcceptat
FROM competente_posturi cp
INNER JOIN posturi p ON cp.IdPost = p.IdPost
INNER JOIN competente c ON cp.IdCompFrunza = c.IdCompetenta
WHERE DenPost = 'Dezvoltator aplicatii - 2'
```

iar rezultatul poate fi cel din figura 8.11.

DENCOMPETENTA	UM	MODEVALUARE	NIVELMINIMACCEPTAT
Java	puncte	test scris	9
SQL	puncte	test scris	7
PL/SQL	puncte	test scris	9
lucru in echipa	puncte	superior	8

Figura 8.11. Competenţele necesare postului *Dezvoltator aplicaţii – 2*

Să trecem la o interogare ceva mai interesantă: Care sunt cei mai indicaţi angajaţi pentru postul *Dezvoltator aplicaţii – 2*? Mai întâi, ne permitem o scurtă digresiune pentru a afişa toate competenţele necesare acestui post şi punctajele fiecărui angajat evaluat la aceleaşi competenţe ca ale postului:

```
SELECT DenPost, DenCompetenta, UM,
  NivelMinimAcceptat AS Punctaj_Minim,
  NumePren, Nivel AS NivelAngajat
FROM competente_posturi cpo
```

```
    INNER JOIN posturi p ON cpo.IdPost = p.IdPost
    INNER JOIN competente c
      ON cpo.IdCompFrunza = c.IdCompetenta
    LEFT OUTER JOIN competente_personal cpe
      ON cpo.IdCompFrunza = cpe.IdCompFrunza
    INNER JOIN personal9 p9 ON cpe.marca=p9.marca
WHERE DenPost = 'Dezvoltator aplicatii - 2'
ORDER BY DenPost, DenCompetenta
```

Rezultatul e destul de stufos (vezi figura 8.12), dar ne ajută să înțelegem mai bine ce avem de făcut în continuare.

DENPOST	DENCOMPETENTA	UM	PUNCTAJ_MINIM	NUMEPREN	NIVELANGAJAT
Dezvoltator aplicatii - 2	Java	puncte	9	Stroe Ionut	9
Dezvoltator aplicatii - 2	Java	puncte	9	Babias Vasile	6
Dezvoltator aplicatii - 2	Java	puncte	9	Lemnaru Mihai	6
Dezvoltator aplicatii - 2	Java	puncte	9	Butuca George	5
Dezvoltator aplicatii - 2	Java	puncte	9	Slavici Minodora	5
Dezvoltator aplicatii - 2	Java	puncte	9	Fodor Iuliana	6
Dezvoltator aplicatii - 2	Java	puncte	9	Slabu Florin	7
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Stroe Ionut	7
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Babias Vasile	9
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Lemnaru Mihai	9
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Butuca George	5
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Fodor Iuliana	7
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Slabu Florin	9
Dezvoltator aplicatii - 2	SQL	puncte	7	Stroe Ionut	7
Dezvoltator aplicatii - 2	SQL	puncte	7	Babias Vasile	5
Dezvoltator aplicatii - 2	SQL	puncte	7	Butuca George	6
Dezvoltator aplicatii - 2	SQL	puncte	7	Dediu Cristina	10
Dezvoltator aplicatii - 2	SQL	puncte	7	Fodor Iuliana	6
Dezvoltator aplicatii - 2	SQL	puncte	7	Slabu Florin	4
Dezvoltator aplicatii - 2	SQL	puncte	7	Slavici Minodora	4
Dezvoltator aplicatii - 2	SQL	puncte	7	Lemnaru Mihai	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Vasilescu Georgeta	7
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Strat Costel	7
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Fodor Iuliana	5
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Slabu Florin	3
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Butuca George	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Lemnaru Mihai	3
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Babias Vasile	5
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Mihuleac Iulian	9
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Stroe Ionut	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Irimia Magdalena	9
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Baboi Cosmina	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Ailenei Jenica	3
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Cosmovici Ionel	2

Figura 8.12. Evaluările angajaților la competențele necesare postului *Dezvoltator aplicații – 2*

Pentru a obține rezultatul corect, trebuie să ținem cont de restricția: dacă un angajat nu are nivelul competenței peste cel minim acceptat al postului respectiv, atunci adună zero puncte pentru acea competență:

```
SELECT NumePren, SUM(CASE WHEN Nivel >= NivelMinimAcceptat
   THEN Nivel ELSE 0 END) AS Punctaj
FROM competente_posturi cpo
   INNER JOIN posturi p ON cpo.IdPost = p.IdPost
```

```
   LEFT OUTER JOIN competente_personal cpe
     ON cpo.IdCompFrunza = cpe.IdCompFrunza
   INNER JOIN personal9 p9 ON cpe.marca=p9.marca
WHERE DenPost = 'Dezvoltator aplicatii - 2'
GROUP BY NumePren
ORDER BY Punctaj DESC
```

Rezultatul este un top al celor mai „apropiaţi", din punctul de vedere al competenţelor solicitate, de postul *Dezvoltator aplicaţii – 2*, vezi figura 8.13.

NUMEPREN	PUNCTAJ
Stroe Ionut	26
Lemnaru Mihai	19
Baboi Cosmina	10
Butuca George	10
Dediu Cristina	10
Babias Vasile	9
Mihuleac Iulian	9
Slabu Florin	9
Irimia Magdalena	9
Ailenei Jenica	0
Slavici Minodora	0
Strat Costel	0
Vasilescu Georgeta	0
Cosmovici Ionel	0
Fodor Iuliana	0

Figura 8.13. Topul celor mai nimeriţi angajaţi pentru postul *Dezvoltator aplicaţii – 2*

Iar ca să ne continuăm *crescendo*-ul, încercăm să răspundem la întrebarea: care sunt cei mai competenţi oameni ai firmei pe locurile pe care sunt angajaţi? Interogarea este cel puţin interesantă (vezi *listing*-ul 8.7 pe Web):

```
SELECT NumePren, Punctaj_pe_post, T1.IdPost
FROM
   (SELECT  NumePren, IdPost, SUM(Nivel) AS Punctaj_pe_post
    FROM
      (SELECT NumePren, marca, p9.idpost, IdCompFrunza,
        NivelMinimAcceptat, Nivel
       FROM personal9 p9 INNER JOIN competente_posturi cpo
        ON p9.IdPost=cpo.IdPost
        INNER JOIN competente_personal cpe
          ON p9.marca=cpe.marca
          AND cpo.IdCompFrunza=cpe.IdCompFrunza
          AND NivelMinimAcceptat <= Nivel
   )
GROUP BY numepren, IdPost) T1 INNER JOIN
   (SELECT IdPost, MAX(Puncte) AS Pct_MAX
```

```
FROM
  (SELECT  IdPost, NumePren, SUM(Nivel) AS Puncte
   FROM
     (SELECT NumePren, marca, p9.idpost,
        IdCompFrunza, NivelMinimAcceptat, Nivel
      FROM personal9 p9 INNER JOIN
        competente_posturi cpo
        ON p9.IdPost=cpo.IdPost
        INNER JOIN competente_personal cpe
        ON p9.marca=cpe.marca
        AND cpo.IdCompFrunza=cpe.IdCompFrunza AND
        NivelMinimAcceptat <= Nivel
     )
    GROUP BY IdPost, NumePren)
GROUP BY IdPost
) T2 ON T1.IdPost=T2.IdPost AND Punctaj_pe_post =  Pct_MAX
```

Interpretarea rezultatului din figura 8.14 este că, în cazul a două posturi (1021 şi 1022), există doi angajaţi cu acelaşi punctaj (maxim) pe post.

NUMEPREN	PUNCTAJ_PE_POST	IDPOST
Cosmovici Ionel	28	1001
Baboi Cosmina	37	1002
Babias Vasile	19	1021
Lemnaru Mihai	19	1021
Butuca George	18	1022
Dediu Cristina	18	1022
Fodor Iuliana	19	1023

Figura 8.14. Cei mai buni pe postul lor

Dacă în interogare înlocuim funcţia `MAX` cu `MIN` (vezi literele îngroşate), obţinem o informaţie cu mult mai interesantă - angajaţii cei mai incompetenţi!

De fapt, în cazul ierarhiilor aşa-zis „generalizate", adică nelimitate ca număr de niveluri, problema stocării este una rezonabilă. Dificultatea cea mai mare ţine de navigarea între niveluri, parcurgerea tuturor descendenţilor unui nod, dată fiind recursivitatea operaţiei. Opţiunile puse la dispoziţie de SGBD-urile actuale sunt relativ izolate, chiar dacă standardul SQL:1999 conţine clauze dedicate tocmai lucrului cu ierarhii (`WITH RECURSIVE... SELECT...`).

Oracle este unul dintre primele produse care au implementat opţiuni pentru obţinerea de informaţii din ierarhii. Cele două clauze principale sunt `START WITH`, prin care se stabileşte linia (linile) „rădăcină" a rezultatului, adică linia/liniile de la care începe căutarea descendenţilor, şi `CONNECT BY [PRIOR]`, pentru specificarea modului în care se construieşte ierarhia. Astfel, pentru a afla competenţele incluse în categoria *baze de date*, este suficientă interogarea:

```
SELECT *
FROM competente
START WITH DenCompetenta='baze de date'
CONNECT BY PRIOR IdCompetenta=IdCompSuperioara
```

rezultatul fiind de genul celui din figura 8.15.

IDCOMPETENTA	DENCOMPETENTA	UM	MODEVALUARE	IDCOMPSUPERIOARA
100017	baze de date			100008
100018	SQL	puncte	test scris	100017
100036	SQL-92	puncte	test scris	100018
100037	SQL_VECHE_	puncte	test scris	100018
100019	Oracle	puncte	test scris	100017
100020	PL/SQL	puncte	test scris	100019
100021	administrare	puncte	test Oracle	100019
100022	developer	puncte	test scris	100019
100023	SGBD open source	puncte	test scris	100017

Figura 8.15. Competenţele de tip *baze de date*

Cum modul de afişare a ierarhiei nu este dintre cele mai clare, tot în Oracle putem folosi opţiunea `LEVEL` (însoţită de funcţia `LPAD`) ce indică nivelul „de adâncime" al descendentului:

```
SELECT IdCompetenta, LPAD('*', 7 * LEVEL, '-')
  || DenCompetenta AS Denumire, UM, ModEvaluare
FROM competente
START WITH DenCompetenta='baze de date'
CONNECT BY PRIOR IdCompetenta=IdCompSuperioara
```

Ceea ce se obţine (vezi figura 8.16) este mult mai uşor de înţeles.

IDCOMPETENTA	DENUMIRE	UM	MODEVALUARE
100017	------*baze de date		
100018	----------*SQL	puncte	test scris
100036	----------*SQL-92	puncte	test scris
100037	----------*SQL_VECHE_	puncte	test scris
100019	----------*Oracle	puncte	test scris
100020	----------*PL/SQL	puncte	test scris
100021	----------*administrare	puncte	test Oracle
100022	----------*developer	puncte	test scris
100023	----------*SGBD open source	puncte	test scris

Figura 8.16. Afişarea arborescentă a competenţelor de tip *baze de date*

La întrebarea care sunt competenţele elementare de tip *baze de date*?, soluţia Oracle este banală:

```
SELECT *
FROM competente
```

```
WHERE IdCompetenta IN
   (SELECT IdCompFrunza FROM competente_elementare)
START WITH DenCompetenta='baze de date'
CONNECT BY PRIOR IdCompetenta=IdCompSuperioara
```

O altă categorie importantă de informații privește nivelul unei anumite competențe sau familii de competențe pe ansamblul organizației. În funcție de datele furnizate de aceste interogări, firma poate lua decizia organizării unor cursuri de „actualizare" sau trimiterii personalului la programe gen master/postuniversitare, cursuri de specializare (Microsoft, Oracle, Cisco etc.). Astfel, interesează la care dintre competențele de tip *baze de date* angajații au cele mai slabe punctaje. Iată și interogarea ce furnizează în Oracle răspunsul:

```
SELECT DenCompetenta, TRUNC(AVG(Nivel),3) Punctaj_Mediu
FROM
   (SELECT *
    FROM competente
    WHERE IdCompetenta IN
      (SELECT IdCompFrunza FROM competente_elementare)
    START WITH DenCompetenta='baze de date'
    CONNECT BY PRIOR IdCompetenta=IdCompSuperioara)
     COMP_ELEM_BD INNER JOIN competente_personal cpe
       ON COMP_ELEM_BD.IdCompetenta=cpe.IdCompFrunza
GROUP BY DenCompetenta
ORDER BY Punctaj_Mediu ASC
```

Rezultatul din figura 8.17 indică un oarecare deficit de SQL, ceea ce înseamnă că unii autori (cel puțin unul) nu și-au dozat prea bine efortul...

DENCOMPETENTA	PUNCTAJ_MEDIU
SQL	6.5
administrare	7.333
PL/SQL	7.666

Figura 8.17. Competențele de tip *baze de date* cele mai deficitare

8.3. Grafuri

Problema grafurilor este una extrem de generoasă și complexă, cu puternice ingrediente matematice și aplicații dintre cele mai diverse. Tratarea noastră va fi însă una simplistă, chiar penibilă pentru cei riguroși. Singurul avantaj ține doar de gradul (îndoielnic) de aplicabilitate a exemplelor discutate.

8.3.1. *Distanţe între localităţi. Trasee*

Începem prin a imagina problema unei firme de transport intern de persoane care acoperă o zonă geografică oarecare din ţara noastră, după cum credem că v-au devenit familiare zecile de „maxi-taxi"-uri care cutreieră şoselele noastre în ultimii ani. Figura 8.18 ilustrează localităţile „deservite" de firma cu pricina. La urma urmei, este de presuspus că acest gen de graf este folosit şi de aplicaţiile „rutiere" care ne indică variantele de drum între două localităţi europene sau de aiurea. Prin comparaţie cu aplicaţiile serioase, graful nostru nu indică şi categoria drumului (şi, implicit, viteza maximă cu care se poate circula), ci numai numărul de kilometri dintre două localităţi. Astfel, variantele de drum pentru traseul Iaşi-Bacău sunt:

- Bacău-Vaslui-Iaşi, care are 85 + 71 = 156 de kilometri;
- Bacău-Roman-Vaslui-Iaşi: 41 + 80 + 71 = 192 km;
- Bacău-Roman-Tg. Frumos-Iaşi: 41 + 40 + 53 = 134 km.

Întâmplarea face ca ruta cea mai scurtă să fie şi cea mai rapidă (cu drumul cel mai bun), însă lucrul acesta este valabil doar uneori.

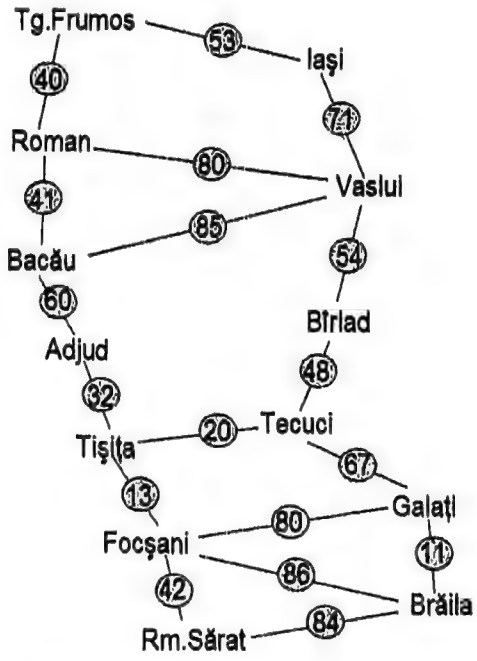


Figura 8.18. Graful câtorva localităţi

Punerea sub formă relaţională a acestor distanţe nu ridică nici un fel de probleme. *Listing*-ul 8.8 conţine scriptul Oracle de creare şi populare a tabelei DISTANŢE.

Listing 8.8. Crearea şi popularea tabelei DISTANŢE

```
DROP TABLE distante ;
CREATE TABLE distante
    (Loc1 VARCHAR2(20),
```

```
    Loc2 VARCHAR2(20),
    Distanta NUMBER(5),
    PRIMARY KEY (Loc1, Loc2)
    ) ;
INSERT INTO distante VALUES ('Iasi', 'Vaslui', 71) ;
INSERT INTO distante VALUES ('Iasi', 'Tg.Frumos', 53) ;
INSERT INTO distante VALUES ('Tg.Frumos', 'Roman', 40) ;
INSERT INTO distante VALUES ('Roman', 'Bacau', 41) ;
INSERT INTO distante VALUES ('Roman', 'Vaslui', 80) ;
INSERT INTO distante VALUES ('Vaslui', 'Birlad', 54) ;
INSERT INTO distante VALUES ('Birlad', 'Tecuci', 48) ;
INSERT INTO distante VALUES ('Tecuci', 'Tisita', 20) ;
INSERT INTO distante VALUES ('Tisita', 'Focsani', 13) ;
INSERT INTO distante VALUES ('Bacau', 'Adjud', 60) ;
INSERT INTO distante VALUES ('Bacau', 'Vaslui', 85) ;
INSERT INTO distante VALUES ('Adjud', 'Tisita', 32) ;
INSERT INTO distante VALUES ('Focsani', 'Galati', 80) ;
INSERT INTO distante VALUES ('Focsani', 'Braila', 86) ;
INSERT INTO distante VALUES ('Focsani', 'Rm.Sarat', 42) ;
INSERT INTO distante VALUES ('Rm.Sarat', 'Braila', 84) ;
INSERT INTO distante VALUES ('Braila', 'Galati', 11) ;
INSERT INTO distante VALUES ('Tecuci', 'Galati', 67) ;
COMMIT ;
```

Date fiind două localități, interesează, mai întâi, ce rute directe și ocolitoare le unesc. Dacă aveți cumva impresia că e floare la ureche, iată în figura 8.19 cele mai scurte 17 variante de drum care ar fi pentru a ajunge de la Iași la Focșani. Cea mai scurtă rută are 206 km, iar cea mai lungă (dintre cele 17), 468.

```
RUTA                                                                DISTANTA
------------------------------------------------------------------ ----------
*Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani*                            206
*Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani*                    239
*Iasi**Vaslui**Bacau**Adjud**Tisita**Focsani*                              261
*Iasi**Vaslui**Roman**Bacau**Adjud**Tisita**Focsani*                       297
*Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Tisita**Focsani*          308
*Iasi**Vaslui**Birlad**Tecuci**Galati**Focsani*                            320
*Iasi**Vaslui**Birlad**Tecuci**Galati**Braila**Focsani*                    337
*Iasi**Tg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Tisita**Focsani*   354
*Iasi**Tg.Frumos**Roman**Vaslui**Bacau**Adjud**Tisita**Focsani*            363
*Iasi**Vaslui**Birlad**Tecuci**Galati**Braila**Rm.Sarat**Focsani*          377
*Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Tecuci**Galati**Focsani*    393
*Iasi**Vaslui**Bacau**Adjud**Tisita**Tecuci**Galati**Focsani*              415
*Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Galati**Focsani*          422
*Iasi**Vaslui**Bacau**Adjud**Tisita**Tecuci**Galati**Braila**Focsani*      432
*Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Galati**Braila**Focsani*  439
*Iasi**Vaslui**Roman**Bacau**Adjud**Tisita**Tecuci**Galati**Focsani*       451
*Iasi**Tg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Galati**Focsani*   468
```

Figura 8.19. Câteva variante de ajuns de la Iași la Focșani

La drept vorbind, mai putem adăuga acestora și alte rute, mai ocolitoare, însă exemplul este suficient de relevant. Dificultatea problemei ține, și în acest caz, nu de structura de stocare (banală, de altminteri), ci de modul de parcurgere a grafului. Soluția pe care o discutăm este una „semi-SQL", în sensul că pentru filtrarea

înregistrărilor din tabelele virtuale, corespunzătoare localităţilor de plecare şi sosire, folosim două variabile publice, `loc_init` şi `loc_dest`. Cum Oracle este mediul în care ne-am apucat să implementăm exemplele noastre, iar în Oracle, într-o frază SQL nu pot fi folosite direct variabile, ci funcţii (ce returnează tocmai valorile variabilelor publice), am recurs la pachetul PAC_LOC – vezi *listing*-ul 8.9.

Listing 8.9. Pachetul şi tabelele virtuale pentru trasee

```
CREATE OR REPLACE PACKAGE pac_loc AS
 loc_init distante.loc1%TYPE := 'Iasi';
 loc_dest distante.loc1%TYPE := 'Focsani' ;
 FUNCTION f_loc_init RETURN loc_init%TYPE ;
 FUNCTION f_loc_dest RETURN loc_init%TYPE ;
END ;
/
-----------------------------------------
CREATE OR REPLACE PACKAGE BODY pac_loc AS
FUNCTION f_loc_init RETURN loc_init%TYPE IS
BEGIN
 RETURN pac_loc.loc_init ;
END ;

FUNCTION f_loc_dest RETURN loc_init%TYPE IS
BEGIN
 RETURN pac_loc.loc_dest ;
END ;
END ;
/
 ---------------------
DROP VIEW v_ordin1 ;
CREATE  VIEW v_ordin1 AS
 SELECT loc1, loc2, distanta, '*'|| loc1 || '**' || loc2 ||'*' AS sir
 FROM distante
 WHERE loc1=pac_loc.f_loc_init()
 UNION
 SELECT loc2, loc1, distanta, '*'|| loc2 || '**' || loc1 ||'*' AS sir
 FROM distante
 WHERE loc2=pac_loc.f_loc_init()
WITH READ ONLY ;
--------------------------
DROP VIEW v_ordin2 ;
CREATE  VIEW v_ordin2 AS
SELECT v.loc1, v.loc2, d.loc2 AS loc3, v.distanta + d.distanta AS distanta,
 sir ||'*'|| d.loc2 ||'*' AS sir
FROM v_ordin1 v INNER JOIN distante d ON v.loc2=d.loc1
WHERE INSTR(sir, '*'||d.loc2||'*') = 0 AND v.loc2 <> pac_loc.f_loc_dest()
UNION
SELECT v.loc1, v.loc2, d.loc1, v.distanta + d.distanta, sir ||'*'|| d.loc1 ||'*' AS sir
FROM v_ordin1 v INNER JOIN distante d ON v.loc2=d.loc2
WHERE INSTR(sir, '*'||d.loc1||'*') = 0 AND v.loc2 <> pac_loc.f_loc_dest()
```

```
WITH READ ONLY ;
--------------------

--...

--------------------
DROP VIEW v_ordin9 ;
CREATE  VIEW v_ordin9 AS SELECT v.loc1, v.loc2, v.loc3, v.loc4, v.loc5, v.loc6, v.loc7, v.loc8,
 v.loc9, d.loc2 AS loc10, v.distanta + d.distanta AS distanta, sir ||'*'|| d.loc2 ||'*' AS sir
FROM v_ordin8 v INNER JOIN distante d ON v.loc9=d.loc1
WHERE INSTR(sir, '*'||d.loc2||'*') = 0 AND v.loc9 <> pac_loc.f_loc_dest()
UNION
SELECT v.loc1, v.loc2, v.loc3, v.loc4, v.loc5, v.loc6, v.loc7, v.loc8, v.loc9, d.loc1,
    v.distanta + d.distanta, sir ||'*'|| d.loc1 ||'*' AS sir
FROM v_ordin8 v INNER JOIN distante d ON v.loc9=d.loc2
WHERE INSTR(sir, '*'||d.loc1||'*') = 0 AND v.loc9 <> pac_loc.f_loc_dest()
WITH READ ONLY ;

DROP VIEW trasee ;
CREATE VIEW trasee AS
 SELECT sir, distanta FROM v_ordin1
    WHERE loc1= pac_loc.f_loc_init() AND loc2=pac_loc.f_loc_dest()
 UNION
    ...
 SELECT sir, distanta FROM v_ordin9
    WHERE loc1= pac_loc.f_loc_init() AND loc10=pac_loc.f_loc_dest()
WITH READ ONLY ;
```

Soluția conținută în *listing* nu este din cale-afară de elegantă, bazându-se pe tabele virtuale stratificate prin care se construiesc rutele. Astfel, V_ORDIN1 conține

```
SQL> SELECT * FROM v_ordin1;

LOC1       LOC2          DISTANTA SIR
---------- ---------- ---------- ----------------------------------------
Iasi       Tg.Frumos          53 *Iasi**Tg.Frumos*
Iasi       Vaslui             71 *Iasi**Vaslui*

SQL>  SELECT * FROM v_ordin2;

LOC1       LOC2       LOC3          DISTANTA SIR
---------- ---------- ---------- ---------- ----------------------------------------
Iasi       Tg.Frumos  Roman              93 *Iasi**Tg.Frumos**Roman*
Iasi       Vaslui     Bacau             156 *Iasi**Vaslui**Bacau*
Iasi       Vaslui     Birlad            125 *Iasi**Vaslui**Birlad*
Iasi       Vaslui     Roman             151 *Iasi**Vaslui**Roman*

SQL> SELECT * FROM v_ordin3;

LOC1       LOC2       LOC3       LOC4          DISTANTA SIR
---------- ---------- ---------- ---------- ---------- ------------------------------
Iasi       Tg.Frumos  Roman      Bacau             194 *Iasi**Tg.Frumos**Roman**Bacau*
Iasi       Tg.Frumos  Roman      Vaslui            173 *Iasi**Tg.Frumos**Roman**Vaslui*
Iasi       Vaslui     Bacau      Adjud             216 *Iasi**Vaslui**Bacau**Adjud*
Iasi       Vaslui     Bacau      Roman             197 *Iasi**Vaslui**Bacau**Roman*
Iasi       Vaslui     Birlad     Tecuci            178 *Iasi**Vaslui**Birlad**Tecuci*
Iasi       Vaslui     Roman      Bacau             192 *Iasi**Vaslui**Roman**Bacau*
Iasi       Vaslui     Roman      Tg.Frumos         191 *Iasi**Vaslui**Roman**Tg.Frumos*
```

Figura 8.20. Conținutul primelor trei tabele virtuale în cazul rutei Iași-Focșani

ruta directă dintre cele două localităţi (dacă există) sau localităţile cu care localitatea de plecare se învecinează. Cum de la Iaşi la Focşani nu este nici un drum direct, cele două linii din exemplul nostru se refereră la distanţele dintre Iaşi şi cele două localităţi învecinate, Tg. Frumos şi Vaslui – vezi figura 8.20.

A doua tabelă virtuală – V_ORDIN2 – uneşte vecinii Iaşului cu vecinii vecinilor Iaşului ş.a.m.d., iar tabela virtuală TRASEE obţine primele *n* variante, directe şi mai ales ocolitoare, dintre cele două localităţi.

8.3.2. *Generarea traseelor*

Dacă pentru turişti discuţia de mai sus poate prezenta interes în alegerea unei rute optime dintre două localităţi (deşi un element fundamental în luarea deciziei, şi care ne lipseşte, este categoria de drum şi calitatea sa), pentru firma de transport intern de călători interesul este de a obţine traseele pe care se vor deplasa vehiculele proprii. De aceea, vom salva rutele care interesează sub sub forma a două tabele, RUTE, ce conţine informaţii generale despre punctele terminus ale unui traseu şi distanţa totală dintre ele, şi LOC_RUTE, adică prin (sau pe lângă) ce localităţi trece traseul respectiv. Fiecare rută va fi identificată printr-un atribut de tip cheie surogat, drept care folosim şi o secvenţă – `seq_idruta`. Prin clauza ON DELETE CASCADE ne asigurăm că, la ştergerea unei rute (care e prea ocolitoare sau, oricum, inacceptabilă), se vor şterge toate localităţile de pe traseul respectiv. *Listing*-ul 8.10 mai conţine şi declanşatoarele de inserare.

Listing 8.10. Secvenţa, două tabele şi declanşatoare pentru generarea rutelor

```
DROP SEQUENCE seq_idruta ;
CREATE SEQUENCE seq_idruta START WITH 1 MINVALUE 1
    MAXVALUE 555555 ORDER NOCYCLE NOCACHE INCREMENT BY 1;

DROP TABLE loc_rute ;
DROP TABLE rute ;
CREATE TABLE rute (
    idruta NUMBER(6) NOT NULL PRIMARY KEY,
    data_generarii DATE DEFAULT CURRENT_DATE NOT NULL,
    loc_init VARCHAR(20),
    loc_dest VARCHAR(20),
    sir VARCHAR2(1000)  );

CREATE TABLE loc_rute (
    idruta NUMBER(6) NOT NULL REFERENCES rute (idruta) ON DELETE CASCADE,
    loc_nr NUMBER(4) NOT NULL,
    loc VARCHAR(20) NOT NULL );

CREATE OR REPLACE TRIGGER trg_rute_ins BEFORE INSERT ON rute FOR EACH ROW
BEGIN
 SELECT seq_idruta.NextVal INTO :NEW.idruta FROM dual ;
END ;
```

```
/
CREATE OR REPLACE TRIGGER trg_loc_rute_ins BEFORE INSERT ON loc_rute
FOR EACH ROW
DECLARE
    v_ultim_nrloc NUMBER(5) := 0 ;
BEGIN
    SELECT MAX(loc_nr) INTO v_ultim_nrloc FROM loc_rute WHERE idruta = :NEW.idruta ;
    IF NVL(v_ultim_nrloc,0) = 0 THEN
        :NEW.loc_nr := 1 ;
    ELSE
        :NEW.loc_nr := v_ultim_nrloc + 1 ;
    END IF ;
END ;
```

Declanşatorul tabelei RUTE este unul banal, vizând atribuirea următoarei valori din secvenţă atributului cheie surogat – `IdRuta`. Nici cel al tabelei LOC_RUTE nu este din cale-afară de complicat, vizând doar gestionarea corectă a atributului care indică a câta localitate de pe un traseu este localitatea de pe linia curentă.

Listing-ul 8.11 este dedicat procedurii PL/SQL prin care se generează cele mai importante rute posibile dintre două localităţi la alegere, localităţi ce sunt tocmai parametrii procedurii. Pe baza valorilor primite drept parametri de intrare, procedura modifică valoarea celor două variabile publice, apoi parcurge fiecare dintre tabelele virtuale „stratificate" – V_ORDIN1, ..., V_ORDIN*n*, generând câte o rută (inserând o linie în RUTE) pentru fiecare linie a unei tabele virtuale în care localităţile de plecare şi sosire sunt cele indicate de către parametrii de intrare, iar pentru fiecare rută se introduce câte o linie în LOC_RUTE pentru orice localitate parcursă.

Listing 8.11. Procedura de generare a rutelor între două localităţi

```
CREATE OR REPLACE PROCEDURE p_generare_rute (loc_init VARCHAR2, loc_dest
VARCHAR2) IS
BEGIN
    pac_loc.loc_init := loc_init ;
    pac_loc.loc_dest := loc_dest ;
    FOR rec_view IN (SELECT * FROM v_ordin1
                WHERE loc1= pac_loc.f_loc_init() AND loc2=pac_loc.f_loc_dest() ) LOOP
        INSERT INTO rute (loc_init, loc_dest, sir, distanta)
            VALUES (rec_view.loc1, rec_view.loc2, rec_view.sir, rec_view.distanta) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc1);
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc2);
    END LOOP ;

    FOR rec_view IN (SELECT * FROM v_ordin2
                WHERE loc1= pac_loc.f_loc_init() AND loc3=pac_loc.f_loc_dest() ) LOOP
        INSERT INTO rute (loc_init, loc_dest, sir, distanta)
            VALUES (rec_view.loc1, rec_view.loc3, rec_view.sir, rec_view.distanta) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc1);
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc2);
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc3);
```

```
    END LOOP ;

    --...

    FOR rec_view IN (SELECT * FROM v_ordin9
              WHERE loc1= pac_loc.f_loc_init() AND loc10=pac_loc.f_loc_dest() ) LOOP
       INSERT INTO rute (loc_init, loc_dest, sir, distanta)
          VALUES (rec_view.loc1, rec_view.loc10, rec_view.sir, rec_view.distanta) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc1) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc2) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc3) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc4) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc5) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc6) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc7) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc8) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc9) ;
       INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc10) ;
    END LOOP ;
END ;
```

Pentru ilustrare, figura 8.21 prezintă modul de lansare a procedurii, localitatea de plecare fiind Iaşi, iar cea de sosire Focşani. Pentru simplificarea discuţiei, sunt apoi

```
SQL>  EXECUTE p_generare_rute ('Iasi', 'Focsani')

PL/SQL procedure successfully completed.

SQL>  DELETE FROM rute WHERE distanta > 250 ;

21 rows deleted.

SQL> SELECT idruta, sir, distanta FROM rute ;

 IDRUTA SIR                                                          DISTANTA
------- ---------------------------------------------------------- ----------
     49 *Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani*                    206
     50 *Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani*            239

SQL> SELECT * FROM loc_rute ;

 IDRUTA     LOC_NR LOC
------- ---------- --------------------
     49          1 Iasi
     49          2 Vaslui
     49          3 Birlad
     49          4 Tecuci
     49          5 Tisita
     49          6 Focsani
     50          1 Iasi
     50          2 Tg.Frumos
     50          3 Roman
     50          4 Bacau
     50          5 Adjud
     50          6 Tisita
     50          7 Focsani
```

Figura 8.21. Conţinutul tabelelor RUTE şi LOC_RUTE pentru cele două trasee alese între Iaşi şi Focşani

eliminate toate traseele mai lungi de 250 Km, rămânând (după cum se vede în frazele SELECT ce urmează comenzii DELETE) doar două rute, una prin Vaslui-Bîrlad--Tecuci-Tişiţa şi cealaltă prin Tg. Frumos-Roman-Bacău-Adjud. Întrucât la declararea restricţiei referenţiale a fost folosită clauza ON DELETE CASCADE, ştergerea liniilor din RUTE va fi însoţită de ştergerea tuturor liniilor-copil. Ultimul SELECT indică tocmai modul de parcurgere a localităţilor pe cele două rute.

8.3.3. *Rezervări şi gestiunea călătoriilor*

Pe drept cuvânt, veţi spune că lucrurile pe care le-am discutat în ultimele două paragrafe au prea puţin în comun cu proiectarea bazelor de date. Nu vă contrazicem (prea vehement). O dată ce am prezentat o soluţie de „cutreierat" grafurile (deşi nu ne-a cerut-o nimeni), trecem la fondul problemei. Firma şi-a stabilit traseele pe care le va acoperi. Fiecare traseu, numit mai sugestiv *rută*, va fi identificat prin celebrul atribut `IdRută`, iar dependenţele pot fi scrise astfel:

- `IdRută` ⟶ `DenRută`, `IdRută` ⟶ `LocPlecare`, `IdRută` ⟶ `Destinaţie`;
- (`IdRută`, `LocalitNr`) ⟶ `Loc`;
- (`Loc1`, `Loc2`) ⟶ `Distanţă`;
- `Localitate` ⟶ `Judeţ`;
- `Localitate` ⟶ `Obs` (aici putem indica dacă se merge pe centură sau se intră în oraş sau alte detalii despre localitate).

Pentru fiecare rută, se stabileşte orarul plecărilor: momentul plecării (`OraPlecare`), dacă circulă sâmbăta sau duminica (`CirculăSD`) şi tipul de vehicul alocat, fiecare tip de vehicul având un anumit număr de locuri disponibile:

- `IdPlecare` ⟶ `IdRuta`;
- `IdPlecare` ⟶ `OrăPlecare`;
- `IdPlecare` ⟶ `TipAuto`;
- `TipAuto` ⟶ `NrLocuri`.

Cursele efective sunt desemnate în limbaj curent în genul *cursa de la 9:00 din 10 ianuarie 2005*; în baza de date însă, ele sunt identificate printr-o altă cheie surogat, `IdCursă`:

- `IdCursa` ⟶ `IdPlecare`;
- `IdCursa` ⟶ `DataCursă`;
- `IdCursa` ⟶ `NrAuto` (de exemplu, IS-39-SQL);
- `IdCursa` ⟶ `Şofer`;
- `NrAuto` ⟶ `TipAuto`.

În timp, din cauza condiţiilor meteo nefavorabile sau a lipsei de clienţi, o serie de plecări pot fi suspendate pe o anumită perioadă:

- `IdSuspendare` ⟶ `IdPlecare`;
- `IdSuspendare` ⟶ `DataSuspendare`;
- `IdSuspendare` ⟶ `DataInițială`;
- `IdSuspendare` ⟶ `DataFinală`.

Ei, şi acum vine partea şi mai interesantă: clienţii îşi pot rezerva (telefonic sau pe Web) locuri la anumite curse. Fiecare rezervare este identificată printr-o cheie surogat (`IdRezervare`), iar informaţiile care interesează sunt: data şi ora la care s-a făcut rezervarea (`DataOraRez`), pe ce nume a fost făcută rezervarea (`NumeRez`), telefonul la care se poate cere confirmarea (`TelRez`), numărul de bilete (locuri) solicitate (`NrBilete`) şi cele două localităţi, de urcare şi coborâre (`DeLa`, `PânăLa`):

- `IdRezervare` ⟶ `DataOraRez`;
- `IdRezervare` ⟶ `NumeRez`;
- `IdRezervare` ⟶ `TelRez`;
- `IdRezervare` ⟶ `NrBilete`;
- `IdRezervare` ⟶ `DeLa`;
- `IdRezervare` ⟶ `PânăLa`.

Cu sau fără rezervare, călătorii primesc la urcare (îmbarcare) un bilet, a cărui contravaloare depinde de numărul de kilometri dintre localităţile între care călătoresc, precum şi de tariful pe kilometru corespunzător.

- `IdBilet` ⟶ `DataOraBilet`;
- `IdBilet` ⟶ `SerieNrBilet`;
- `IdBilet` ⟶ `IdRezervare`;
- `IdBilet` ⟶ `DeLa`;
- `IdBilet` ⟶ `PânăLa`;
- `IdBilet` ⟶ `ValBilet`.

Costul pe kilometru se stabileşte pe intervale: de la 0 la 20 Km, tariful pe kilometru este de 1500 lei, de la 20 la 40 - 1450, de la 40 la 70 - 1425 etc.:

- (`KmLimitaInf`, `KmLimitaSup`) ⟶ `CostPeKm`.

Dacă mai punem la socoteală şi pe cele de incluziune, obţinem un graf al dependenţelor precum cel din figura 8.22.

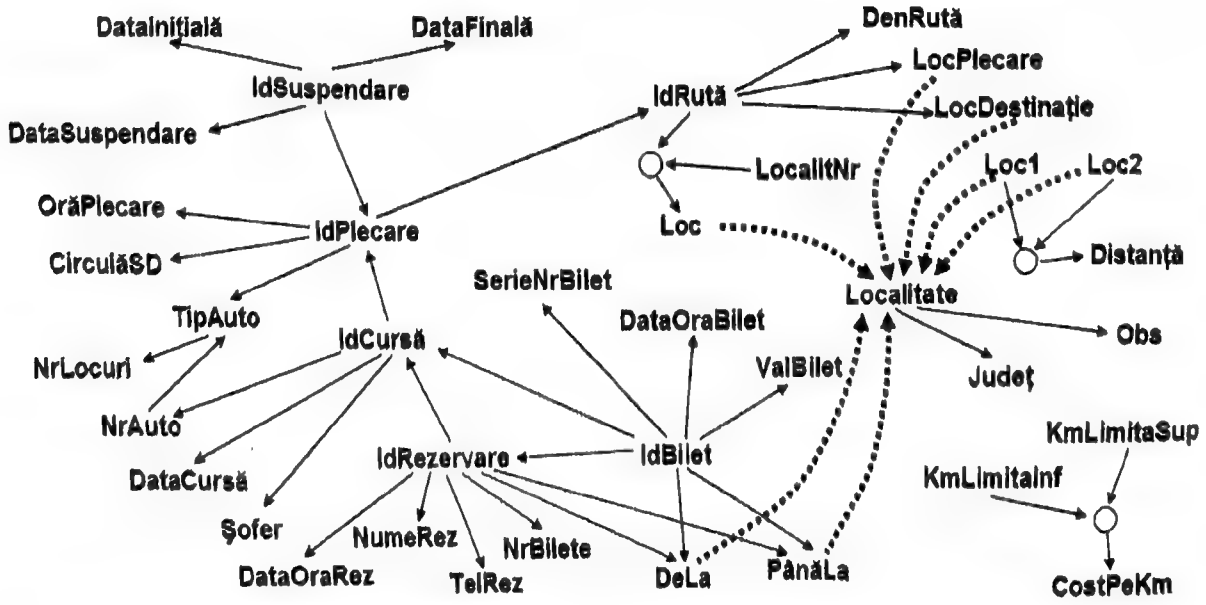


Figura 8.22. Graful dependenţelor pentru baza de date TRANSPORT_INTERN_CĂLĂTORI

Decupând relațiile din graf, obținem tabelele din figura 8.23. Scriptul de creare a tabelelor și de definire a restricțiilor (8.12) poate fi descărcat de pe pagina Web indicată (*http : //www.feaa.uaic.ro/cercetare/publicatii*).

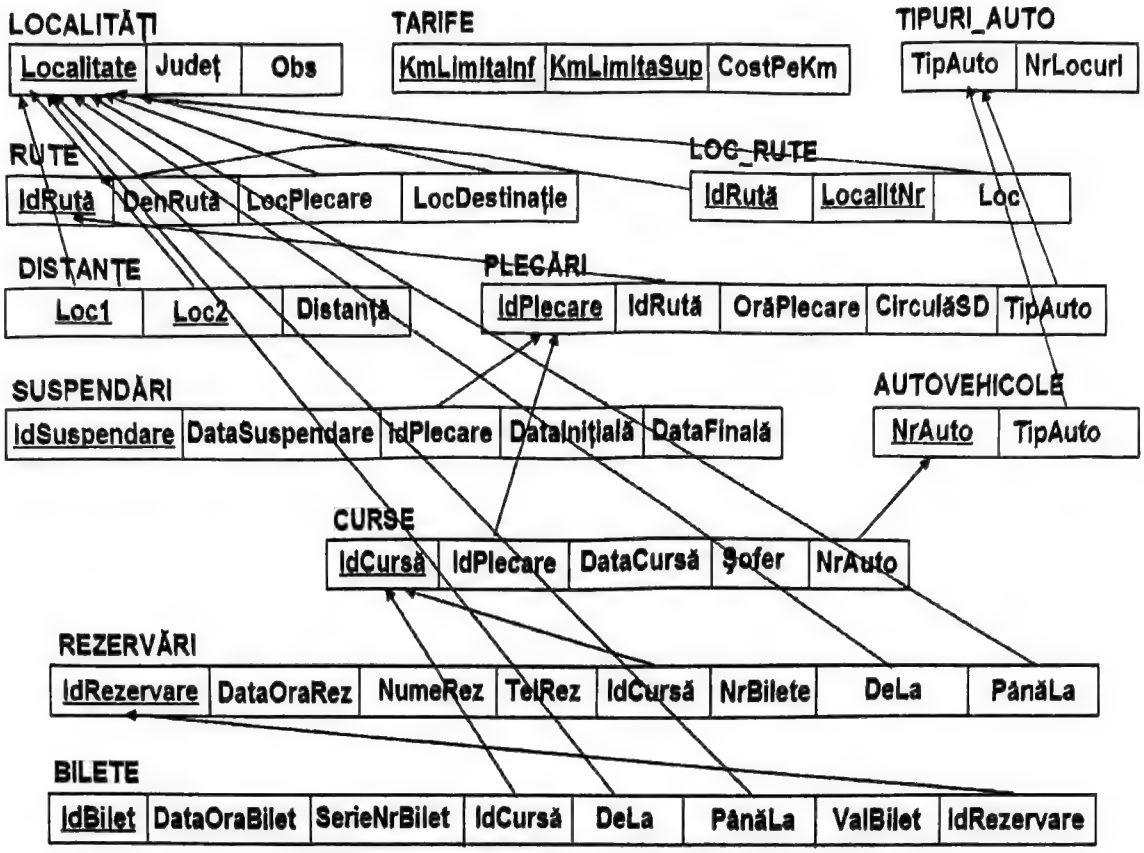


Figura 8.23. Tabelele bazei de date TRANSPORT_INTERN_CĂLĂTORI

Datorită dimensiunilor apreciabile, scriptul de populare a celor mai multe dintre tabele nu a fost inclus în carte, fiind disponibil pe Web la aceeași adresă, sub forma *listing*-ului 8.13. În schimb, putem cheltui câteva cuvinte pe funcția din *listing*-ul 8.14, menită a calcula valorea unui bilet. Funcției i se transmit trei parametri : `vLocUrcare`, indică stația de urcare a pasagerului, `vLocCoborâre`, ce indică stația de coborâre și `vIdRuta` – ruta pe care o urmează autovehiculul cu care călătorește. Pe baza înregistrărilor din tabela LOC_RUTE, se determină a câta localitate de pe rută este cea de urcare (`i_urcare`) și a câta este cea de coborâre (`i_coborâre`). Cu ajutorul cursorului definit prin structura FOR... END LOOP se parcurg toate localitățile dintre urcare și coborâre, adunându-se, de fiecare dată, numărul de kilometri în variabila `v_NrKm`.

Listing 8.14. Funcția pentru calculul prețului unei călătorii

```
CREATE OR REPLACE FUNCTION f_cost_bilet (
    vLocUrcare localitati.Localitate%TYPE,
    vLocCoborire localitati.Localitate%TYPE,
    vIdRuta rute.IdRuta%TYPE ) RETURN NUMBER
IS
    i_urcare NUMBER(3) := 0 ;
    i_coborire NUMBER(3) := 0 ;
```

```
    v_nrkm NUMBER(4) := 0 ;
    v_distanta NUMBER(4) := 0 ;
    v_cost tarife.CostPeKm%TYPE := 0;
    v_loc1 localitati.Localitate%TYPE ;
    v_loc2 localitati.Localitate%TYPE ;
BEGIN
    SELECT LocalitNr INTO i_urcare FROM loc_rute WHERE IdRuta=vIdRuta AND
    loc=vLocUrcare ;
    SELECT LocalitNr INTO i_coborire FROM loc_rute WHERE IdRuta=vIdRuta AND
    loc=vLocCoborire ;

    IF i_urcare >= i_coborire THEN
        RAISE_APPLICATION_ERROR(-20177, 'Ruta/localitati gresite!');
    END IF ;

    FOR rec_loc IN (SELECT * FROM loc_rute WHERE IdRuta=vIdRuta
        AND LocalitNr BETWEEN i_urcare AND i_coborire ORDER BY LocalitNr ) LOOP
        IF rec_loc.LocalitNr = i_urcare THEN
            v_loc1 := rec_loc.Loc ;
        ELSE
            v_loc2 := rec_Loc.Loc ;
            SELECT distanta INTO v_distanta FROM distante
            WHERE loc1 = LEAST(v_loc1, v_loc2) AND
            loc2 = GREATEST(v_loc1, v_loc2) ;
            v_NrKm := v_NrKm + v_distanta ;
            v_loc1 := v_loc2 ;
        END IF ;
    END LOOP ;

    SELECT ROUND(CostPeKm * v_NrKm,-3) INTO v_cost FROM tarife
    WHERE v_NrKm >= KmLimitaInf AND v_NrKm < KmLimitaSup ;

    RETURN v_cost ;
END ;
```

În final, pe baza numărului de kilometri, se calculează contravaloarea biletului prin înmulţirea numărului cu tariful pe kilometru din tranşa corespunzătoare a tabelei TARIFE. Figura 8.24 indică modul de apel în Oracle SQL*Plus a funcţiei pentru determinarea costului biletului dintre Iaşi şi Focşani pe varianta Tg. Frumos-Roman--Adjud (IdRută=4) şi varianta Vaslui-Tecuci (IdRută=3).

Schema de stocare la care am ajuns permite rezolvarea şi a unor probleme ceva mai delicate. Să ne oprim un pic la rezervările *on* sau *offline*. Un posibil călător sună la dispecerat şi declară că vrea două bilete de la Iaşi la Tecuci pentru cursa de la ora 6:00 din data de 2 decembrie 2004. Microbuzul alocat acestei curse are 16 locuri. Problema este că, înainte de a confirma rezervarea, aplicaţia trebuie să verifice câte locuri au fost rezervate nu numai între Iaşi şi Tecuci, ci şi în orice interval care cuprinde cele două localităţi, cum ar fi: Vaslui-Bîrlad, Vaslui-Tecuci, Vaslui-Focşani, Bîrlad-Tecuci etc. O soluţie nu din cale-afară de rapidă ţine de folosirea unei funcţii

```
SQL>  begin
  2   dbms_output.put_line(f_cost_bilet('Iasi', 'Focsani', 4));
  3   end ;
  4  /
311000

PL/SQL procedure successfully completed.

SQL> begin
  2  dbms_output.put_line(f_cost_bilet('Iasi', 'Focsani', 3));
  3  end ;
  4  /
266000

PL/SQL procedure successfully completed.

SQL> |
```

Figura 8.24. Două apeluri ale funcției f_cost_bilet

care să returneze poziția unei localități pe un traseu (rută), funcție folosită de declanșatorul de inserare al tabelei REZERVĂRI – vezi *listing*-ul 8.15.

Listing 8.15. O secvență, o funcție pentru determinarea poziției pe o rută a unei localități și declanșatorul de inserare pentru rezervări

```
DROP SEQUENCE seq_IdRezervare ;
CREATE SEQUENCE seq_IdRezervare START WITH 1 INCREMENT BY 1 MINVALUE 1
 MAXVALUE 99999999 NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE FUNCTION f_ordine_loc (
    Loc_ loc_rute.Loc%TYPE, IdRuta_ loc_rute.IdRuta%TYPE)
    RETURN loc_rute.LocalitNr%TYPE
IS
    v_nr loc_rute.LocalitNr%TYPE ;
BEGIN
    SELECT LocalitNr INTO v_nr FROM loc_rute WHERE IdRuta=IdRuta_ AND loc=Loc_ ;
    RETURN v_nr ;
END ;
/

CREATE OR REPLACE TRIGGER trg_rezervari_ins
    BEFORE INSERT ON rezervari FOR EACH ROW
DECLARE
    v_NrRezervari rezervari.NrBilete%TYPE := 0 ;
    v_NrLocuriAuto rezervari.NrBilete%TYPE := 0 ;
    i_urcare NUMBER(3) := 0 ;
    i_coborire NUMBER(3) := 0 ;
    i NUMBER(3) := 0 ;
    v_IdRuta rute.IdRuta%TYPE ;
BEGIN
    -- In variabila v_NrLocuriAuto se stocheaza capacitatea auto/microbuzului
    SELECT NrLocuri INTO v_NrLocuriAuto FROM tipuri_auto WHERE TipAuto=
```

```
        (SELECT TipAuto FROM autovehicule WHERE NrAuto=
          (SELECT NrAuto FROM curse2 WHERE IdCursa = :NEW.IdCursa)) ;

    -- se determina Id-ul rutei pentru a vedea ce localitati se parcurg
    SELECT IdRuta INTO v_IdRuta
    FROM curse2 c INNER JOIN plecari p ON c.IdPlecare=p.IdPlecare
    WHERE IdCursa=:NEW.IdCursa ;

    -- se obtine numarul localitatii de urcare si al celei de coborire de pe traseu
    i_urcare := f_ordine_loc (:NEW.DeLa, v_IdRuta) ;
    i_coborire :=  f_ordine_loc (:NEW.PinaLa, v_IdRuta) ;
    IF i_urcare >= i_coborire THEN
        RAISE_APPLICATION_ERROR(-20177, 'Ruta/localitati gresite!');
    END IF ;

    -- se parcurge fiecare portiune dintre localitatile de urcare si coborire
    -- si se verifica daca nr. rezervarilor depaseste numarul locurilor din autovehicul
    FOR i IN i_urcare..i_coborire-1 LOOP
        SELECT SUM(NrBilete) INTO v_NrRezervari
        FROM (SELECT * FROM rezervari WHERE IdCursa=:NEW.IdCursa)
        WHERE f_ordine_loc(DeLa, v_IdRuta) <= i AND f_ordine_loc(PinaLa, v_IdRuta)
        >= i + 1 ;

        IF NVL(v_NrRezervari,0) + :NEW.NrBilete >= v_NrLocuriAuto  THEN
            RAISE_APPLICATION_ERROR(-20178, 'Nu mai sunt locuri! ');
        END IF ;
    END LOOP ;

    SELECT seq_IdRezervare.NextVal INTO :NEW.IdRezervare FROM dual ;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20179,
            'Ruta/cursa pentru care se doreste rezervarea nu este introdusa!') ;
END ;
```

În mare, ideea declanşatorului este următoarea : traseul dintre cele două localităţi se rupe pe porţiuni delimitate de localităţile definite în tabela LOC_RUTE, iar pentru fiecare porţiune a traseului se verifică dacă rezervările deja efectuate pentru această cursă, plus numărul locurilor solicitate la rezervarea curentă nu depăşesc numărul locurilor din autovehicul. Soluţia nu pare însă prea rapidă, dacă volumul datelor este imens (după ani şi ani de folosire a aplicaţiei).

8.4. Incluziuni & echivalenţe

Continuăm discuţiile din acest capitol cu o altă problemă marginală. Figura 8.4 conţine structura personalului unei instituţii academice, iar interesul nostru a fost de a ilustra modul în care se poate gestiona o structură ierarhică atunci când ierarhiile

simple se combină cu cele multiple. Extindem un pic problema, gândindu-ne la un mini-mecanism de căutare a cărţilor într-o bibliotecă, librărie etc., căutare derulată după anumite cuvinte-cheie sau, mai elevat spus, *sintagme*. Partea interesantă, dar şi dificilă a discuţiei ţine de faptul că relaţiile în care se află sintagmele unele cu altele sunt, în afară de *independenţă*, nu numai de *incluziune*, simplă şi multiplă, dar şi de *echivalenţă*. Să luăm o porţiune dintr-o „încrengătură" de sintagme - vezi figura 8.25.

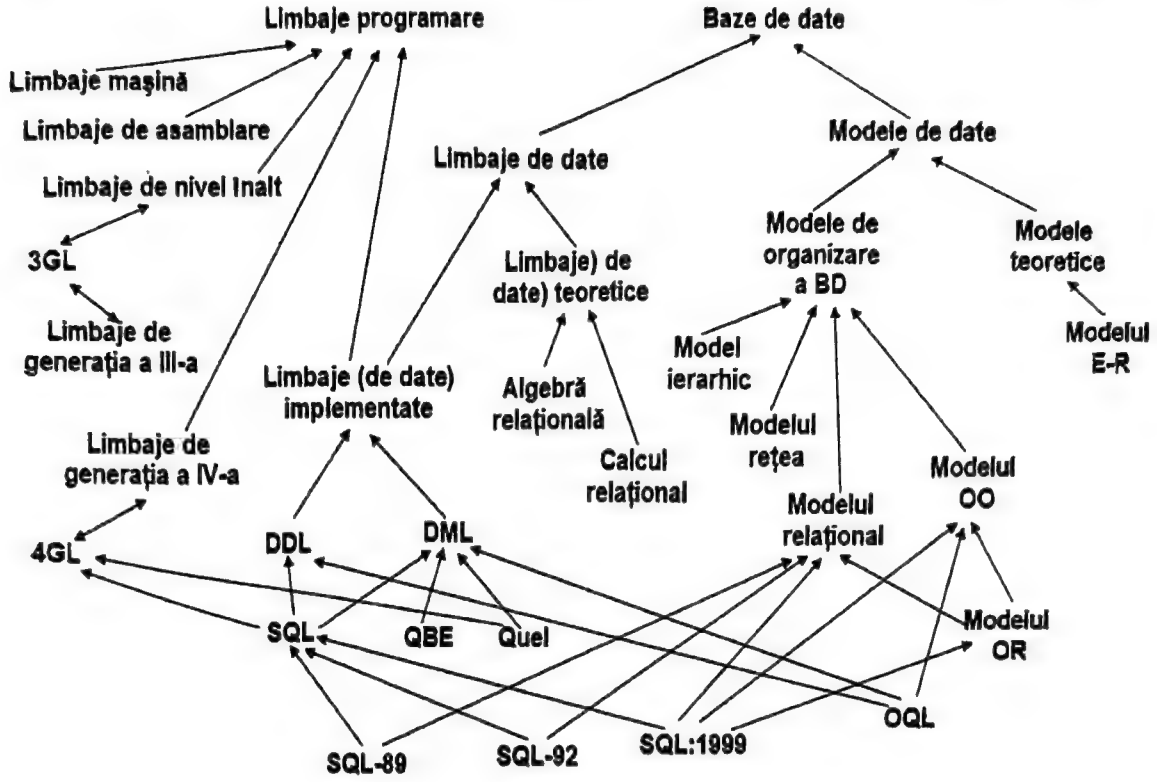


Figura 8.25. Rapoarte de incluziune şi echivalenţă între cuvinte-cheie (sintagme)

Astfel, sintagmele: *limbaje maşină*, *limbaje de asamblare*, *limbaje de nivel înalt*, *limbaje de generaţia a IV-a* şi *limbajele (de date) implementate* sunt toate incluse în categoria *limbaje de programare*. Sintagma *SQL-89* ţine de două noţiuni, *SQL* şi *modelul relaţional*. În plus, sintagma *4GL* este echivalentă cu *limbaje de generaţia a IV-a*, iar *3GL* este echivalentă atât cu *limbajele de nivel înalt*, cât şi cu *limbajele de generaţia a III-a*. Scopul acestei organizări alambicate este ca, în funcţie de modul în care s-au definit cuvintele-cheie pentru cărţile din bibliotecă, la căutarea după o anumită sintagmă să fie indicate nu numai cărţile cu sintagma respectivă, ci şi cele la care au fost definite sintagme echivalente şi sintagme subordonate.

O soluţie de rezolvare a acestei probleme ar putea face apel la un „nomenclator" de cuvinte-cheie (sintagme), apoi, fiecare incluziune, adică relaţie sintagmă copil-sintagmă părinte să fie identificată printr-un atribut de tip cheie surogat (`IdIncluziune`), iar, de asemenea, fiecare echivalenţă dintre două sintagme să fie consemnată prin alocarea unui identificator de acelaşi tip surogat (`IdEchivalenţă`). Astfel, graful dependenţelor funcţionale şi de incluziune ar putea arăta ca în figura 8.26. Atributul `RelevanţăISBN` sugerează cât de bine (mult) este tratat subiectul desemnat

printr-o sintagmă în cartea curentă, valorile acestuia fiind acordate după sistemul de notare de la 1 la 10 sau de la 1 la 5.

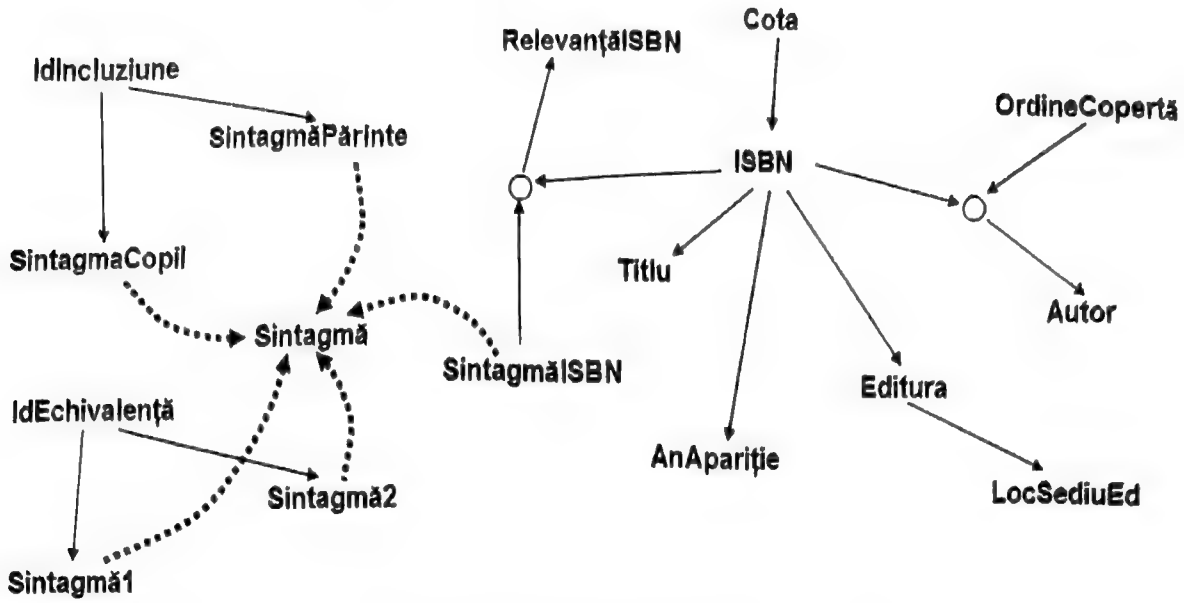


Figura 8.26. Noul graf al dependențelor pentru BD BIBLIOTECĂ

Decuparea relaţiilor din graf este banală, după toată experienţa pe care am acumulat-o – vezi figura 8.27.

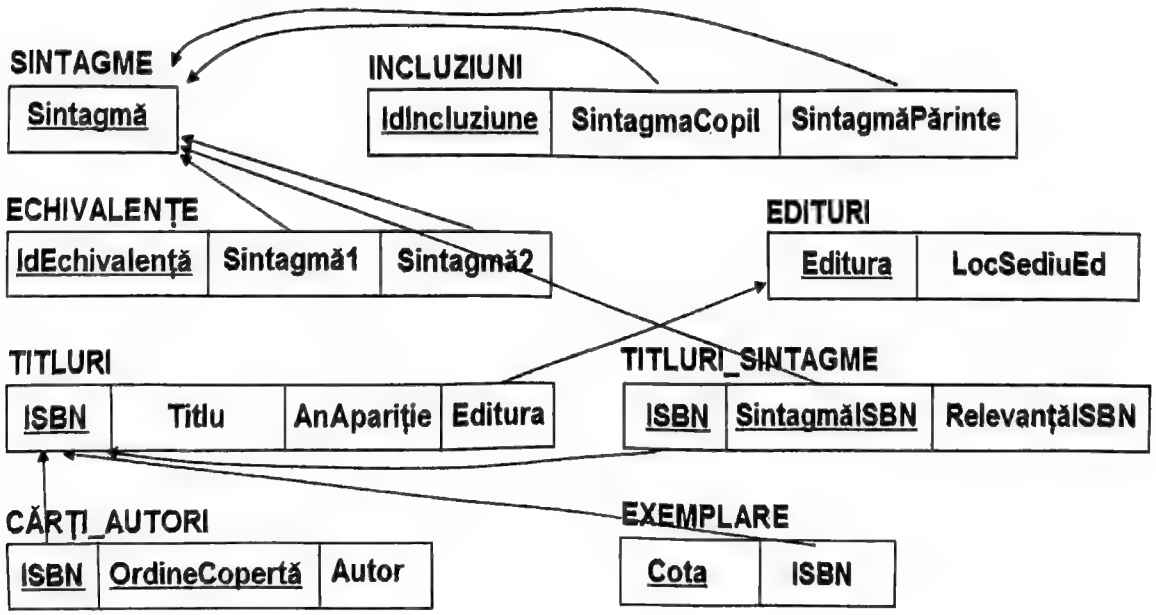


Figura 8.27. Noua schemă a bazei de date BIBLIOTECĂ

O dată pusă la punct problematica stocării relaţiilor de echivalenţă, ca şi în cazul grafurilor, dificultatea este pasată mecanismului de „parcurgere" a ţesăturii de incluziuni şi echivalenţe. Se poate imagina un mecanism la fel de primitiv precum cel folosit la grafuri, de construire a unor tabele virtuale „concentrice" care să furnizeze, pe rând, vecinii sintagmei de plecare, apoi vecinii vecinilor acesteia ş.a.m.d. Noi însă ne oprim aici cu acest capitol, chiar dacă veţi spune că v-am lăsat în mijlocul drumului.

CAPITOLUL 9

Denormalizare şi postnormalizare

Scopul prioritar al proiectării bazei de date este găsirea unei scheme de stocare care să preia toate informaţiile relevante pentru aplicaţie, să permită declararea restricţiilor pe care trebuie să le respecte datele în bază şi să ofere suport pentru prelucrările ulterioare (grupare, însumare, analize încrucişate) ale datelor în vederea identificării informaţiilor relevante pentru fiecare grup de utilizatori. Cu toate acestea, proiectarea anticipează şi o serie de probleme vitale pentru bunul mers al aplicaţiei, cum ar fi viteza de acces, siguranţa datelor, politica de acces etc. Chiar dacă acestea ţin preponderent de *proiectarea fizică* a bazei, câteva chestiuni pot fi abordate şi în faza de proiectare logică.

Problema denormalizării este una de „tradiţie" în proiectarea bazelor de date relaţionale şi destul de agitată, dacă luăm în considerare diferenţele de opinii în definirea şi mecanismele sale. Definită încă din anii '80 şi disputată în a doua parte a anilor '90, denormalizarea şi-a stabilizat propria-i „piaţă", alcătuită din două tabere ireductibile, una a teoreticienilor de tip Fabian Pascal[1], fundamentalişti gata de a salva oricând teoria dând vina exclusiv pe piaţă, mai precis, pe slăbiciunile SGBD-urilor, iar cealaltă a practicienilor, uneori ignoranţi de-a binelea în materie chiar de noţiuni fundamentale despre bazele de date, pentru care orice, dar absolut orice este posibil şi permis într-o schemă de baze de date, atâta vreme cât binele aplicaţiei sau al programatorilor o cere sau cerşeşte.

Ca şi până acum, noi vom face în acest capitol naveta între fundamentalişti şi ignoranţi, încercând pe cât posibil să conturăm câteva atuuri şi pericole ale denormalizării şi, firește, atrăgându-ne dezaprobarea şi antipatia ambelor categorii prin „întinarea nobilelor idealuri"[2] de o manieră „imparţială".

9.1. Denormalizarea – rău-rău, dar chiar necesar ?

Una dintre primele definiţii ale denormalizării a fost formulată de Candace Fleming şi Barbara von Halle: procesul prin care, după definirea unei structuri de date stabile, deplin normalizate, se introduc selectiv date redundante pentru facilitarea

1. Vezi, spre exemplu, [Pascal 01].
2. Nu intenţionez să plătesc drepturi de autor pentru aceste trei cuvinte lemnoase ale unui personagiu clădit din aceeaşi esenţă (precum cuvintele).

unor cerinţe specifice legate de performanţă[3]. Într-o lucrare mai recentă, Robert Muller definea denormalizarea ca procesul combinării de tabele sau obiecte pentru a creşte viteza de acces, de obicei prin evitarea joncţiunilor[4]. Ceva mai analitică, Toby Teorey definea denormalizarea ca un proces alcătuit din următoarele activităţi[5]:

- selectarea proceselor dominante, pe baza frecvenţei, volumului, priorităţii;
- definirea unor extensii ale tabelelor existente (atribute sau chiar tabele noi) care să amelioreze performanţele de interogare a bazei;
- evaluarea costului total al interogării, stocării şi actualizării;
- evaluarea efectelor colaterale, cum ar fi riscul pierderii integrităţii datelor.

Hoffer ş.a. leagă denormalizarea strict de nivelul fizic al schemei unei baze de date, definind-o destul de exotic drept procesul transformării relaţiilor normalizate în specificaţii privind înregistrări fizice nenormalizate[6]. Din contră, pentru Elmasri şi Navathe, procesul denormalizării se desfăşoară preponderent la nivel logic: procesul stocării joncţiunilor dintre relaţii aflate în forme normale avansate ca relaţii aflate, inevitabil, în forme normale mai „joase" este cunoscut sub numele de denormalizare[7]. Amuzant este şi faptul că, dacă pentru Elmasri şi Navathe denormalizarea este o „cădere" din forme normale înalte în altele mai joase, Connoly şi Begg definesc denormalizarea ca o rafinare a schemei relaţionale[8].

Probabil cel mai des invocat motiv pentru denormalizare este timpul necesar efectuării joncţiunilor între tabelele fragmentate rezultate în urma normalizării. Dacă numărul tabelelor este mare, numărul de rapoarte/informaţii solicitate de utilizatori este uriaş şi presupune joncţiuni numeroase, utilizatorii sunt numeroşi, iar cerinţele de viteză sunt vitale, tentaţia este de a introduce atribute redundante sau de a duplica atribute în tabele diferite, astfel încât numărul joncţiunilor necesare obţinerii informaţiilor frecvente să fie cât mai mic cu putinţă. Prin urmare, uneori, deviaţiile de la o bună normalizare par logice şi practice[9]. Chris Date sintetizează foarte bine argumentul în trei părţi ale denormalizatorilor[10]:

1. Normalizarea deplină înseamnă multe relaţii separate logic.
2. Mai multe relaţii logic separate înseamnă mai multe fişiere de stocare separate fizic.
3. Mai multe fişiere de stocare separate fizic înseamnă un mai mare volum de operaţii de intrare/ieşire, deci consum de resurse.

3. [Fleming & von Halle 89], p. 440.
4. [Muller 99], p. 6.
5. [Teorey 99], p. 8.
6. [Hoffer ş.a. 02], p. 215.
7. [Elmasri & Navathe 00], p. 484.
8. [Connoly & Begg 02], p. 507.
9. [Poole 00].
10. [Date 04], p. 393.

Craig Mullins sugerează că există câteva simptome care trebuie să atragă atenţia asupra schemelor susceptibile de denormalizare[11] :

a) rapoarte şi interogări critice extrag informaţiile prin interogarea simultană a mai multor tabele ;
b) există grupuri repetitive care sunt procesate nu atât individual, cât la nivel de set (grup) ;
c) interogările/rapoartele reclamă calcule aplicabile multor atribute aflate în tabele diferite ;
d) utilizatori diferiţi accesează simultan aceeaşi tabelă sau grup de tabele după criterii diferite ;
e) cheile primare ale unor tabele sunt prea mari şi consumă timp la joncţionări ;
f) există diferenţe majore, din punct de vedere statistic, între gradul de folosire în interogări a unor atribute faţă de altele.

Bock şi Schrage remarcau, pe bună dreptate, că puţine sunt cărţile care prezintă în detaliu cum anume trebuie făcută denormalizarea[12], dând totuşi ca exemplu lucrarea Gillette ş.a.[13], care recomandă trei linii directoare :

1. identificarea aplicaţiilor-program care conţin un număr excesiv de joncţiuni, desemnând drept excesive cazurile în care o tabelă virtuală, tabelă-obiect, necesită mai mult de trei joncţiuni ;
2. reducerea numărului de chei străine, care ar atrage descongestionarea indexurilor ce trebuie gestionate la actualizările tabelelor ; reducerea cheilor străine se traduce, de fapt, prin reducerea numărului de tabele din bază ;
3. menţinerea mecanismului de integritate pentru evitarea anomaliilor datelor din bază.

Cele trei linii directoare nu suferă doar de generalitate, ci chiar sunt discutabile, în sine, întrucât aplicarea lor trebuie raportată permanent la situaţia efectivă „din teren".

După cum punctau Fleming şi von Halle, problemele de performanţă în utilizarea bazelor de date nu ţin de structura relaţională obţinută prin normalizare, ci de mecanismul deficient de stocare şi acces la date specific fiecărui produs comercial. Dacă SGBD-urile ar fi perfecte, atunci schema obţinută în urma proiectării bazei de date nu ar prezenta probleme de performanţă[14]. Ideea este susţinută şi de Bock şi Schrage, pentru care slabele performanţe ale sistemului depind şi de[15] :

- platformele hardware inadecvate ;
- slaba optimizare (acordare – *tuning*) a SGBD-ului ;
- tehnicile de programare rudimentare, stufoase sau neglijente ;
- proiectarea conceptuală/logică inadecvată ;
- schema fizică nepusă bine la punct.

11. [Mullins 98].
12. [Bock & Schrage 02].
13. [Gillette ş.a. 95].
14. [Fleming & von Halle 89], p. 435.
15. [Bock & Schrage 02].

Firește, pentru lucrarea de față ne interesează doar aspectele ce țin de schemele (logică, și în mai mică măsură fizică) bazei.

Chris Date demontează argumentele denormalizării, deoarece acestea pornesc de la premisa falsă că o tabelă corespunde unui fișier fizic. Or, nicăieri în modelul relațional acest lucru nu este nici măcar sugerat. Paradoxul sesizat de autor este că, dacă este necesară, denormalizarea ar putea fi efectuată la nivel fizic, dar la nivelul fizic nu putem discuta despre relații, ci despre fișiere fizice[16]. De fapt, Date (ca și Fabian Pascal, de altminteri) nu scapă prilejul de a da vina pe SQL și actualele SGBD-uri bazate pe SQL care nu operează cum trebuie separarea nivelului logic de cel fizic într-o bază de date.

Fleming și von Halle sunt de părere că deviațiile de la structura obținută prin normalizare sunt permise numai dacă[17]: (1) alte mecanisme de optimizare sunt inadecvate sau indisponibile; (2) cerințele de performanță sunt imperioase; (3) există posibilitatea implementării unui mecanism care, chiar în condițiile duplicării informațiilor, asigură integritatea și consistența datelor; (4) orice deviere de la modelul logic al datelor este documentată și explicată, astfel încât să existe o cunoaștere deplină a artificiilor folosite în schemă. Ideea este susținută și de Craig Mullins, care sugerează ca, înainte de a porni la denormalizarea unei scheme, să se zăbovească asupra a trei chestiuni[18]:

- sistemul poate atinge performanțe acceptabile fără denormalizare?
- după denormalizare, performanța sistemului va continua să fie inacceptabilă?
- va deveni sistemul, în urma denormalizării, mai puțin sigur?

Dacă răspunsul la măcar una dintre aceste întrebări este afirmativ, atunci denormalizarea trebuie evitată.

Chiar și adepții necondiționați ai denormalizării atrag atenția asupra faptului că denormalizarea are un cost, tradus, de cele mai multe ori, în pierderea flexibilității, diminuarea scalabilității, performanțe mai slabe ale aplicației/bazei de date sau chiar amenințări la adresa integrității datelor. Plus efortul necesitat de mecanismele laborioase de păstrare a corectitudinii informațiilor paralele, redundante[19].

9.2. Categorii de operații legate de denormalizare

Există o largă tipologie a operațiunilor ce pot fi subsumate denormalizării. De fapt, după cum spunea și Date, una dintre problemele ușor de conștientizat ține de faptul că, în general, este destul de ușor de identificat momentul în care trebuie începută denormalizarea și aproape imposibil de știut când trebuie oprită[20]. Ironic, autorul plusează afirmând că, dacă în cazul normalizării obiectivul era aducerea bazei într-o

16. [Date04], p. 393
17. [Fleming & von Halle 89], p. 438.
18. [Mullins 98].
19. Vezi, spre exemplu, [Hoberman 02].
20. [Date 04], p. 394.

formă normală cât mai „înaltă", denormalizarea ar putea să-şi propună drept ţel o formă normală cât mai joasă.

Dick Root identifică şapte tehnici utilizabile în denormalizare[21] :

1. date duplicate (copii integrale sau parţiale ale unor atribute sau grupuri de atribute) ;
2. date derivate/calculate ;
3. chei surogat ;
4. supra-normalizare – partiţionare/segmentare pe verticală : ruperea unei relaţii în două sau mai multe ;
5. partiţionare/segmentare pe orizontală : linii diferite dintr-o relaţie sunt stocate în tabele diferite ;
6. joncţiuni stocate – joncţionarea a două sau mai multe tabele şi stocarea rezultatului într-o tabelă de sine stătătoare ;
7. grupuri de date repetitive (date de tip vector).

Într-o cu totul altă abordare, Bock şi Schrage tratează câteva modalităţi de denormalizare aplicabile fiecărei forme normale[22] : 1NF, 2NF, 3NF, cu câteva sugestii pentru formele normale „mai înalte". Acestei abordări i se poate reproşa faptul că amestecă denormalizarea cu normalizarea, în timp ce majoritatea autorilor recomandă ca discuţiile legate de normalizare să fie purtate numai după ce schema bazei de date este adusă în 3NF sau într-o formă ulterioară acesteia.

Mullins recomandă folosirea, în cadrul denormalizării, a următoarelor artificii[23] :

- tabele pre-joncţionate ;
- tabele-raport, construite în funcţie de rapoartele critice ale aplicaţiei ;
- tabele-oglindă, mai ales în mediile cu acces simultan al unui mare număr de utilizatori ;
- tabele partiţionate, în funcţie de grupurile de utilizatori ;
- tabele combinate, pentru consolidarea legăturilor *unu la mai mulţi* într-o singură tabelă ;
- date copiate (duplicate) ;
- grupuri repetitive ;
- date derivate ;
- tabele de mărire a vitezei, recomandabile mai ales în cazul ierarhiilor.

Atributele duplicate sunt, probabil, cea mai comună categorie a denormalizării. Un atribut este duplicat dacă se repetă, sub formă identică sau similară, într-una sau mai multe tabele din bază. Din această categorie sunt excluse atributele de tip cheie străină necesare pentru a pune în legătură relaţiile-copil cu cele părinte. Fleming şi von Halle sunt de părere că există patru tipuri de atribute duplicate şi nici unul nu este absolut necesar[24] :

1. copii exacte ale unui atribut (se exclud din discuţie cheile străine) ;

21. [Root 01].
22. [Bock & Schrage 02].
23. Mullins 98].
24. [Fleming & von Halle 89], p. 440.

2. atribute derivate din cele existente sau sintetice (calculate prin însumare, numărare sau alte calcule aplicate altor atribute);
3. grupuri repetitive care, inițial, apar pe mai multe linii în tabelă;
4. substitute artificiale ale unor atribute existente.

Deși nu o fac la modul cel mai explicit, cele două autoare lasă să se înțeleagă că doar primul tip (copiile exacte în tabele diferite) poate fi încadrat la categoria *denormalizare*. Este și părerea noastră și, în același timp, argumentul pentru tăcerea mormântală de până acum în privința denormalizării, deși redundanțe am introdus încă din capitolul 7.

Țin totuși cheile surogat de denormalizare? Root spune că da, iar dacă interpretăm în sens larg al patrulea tip de atribute duplicate din clasificarea de mai sus, și Fleming și von Halle sunt pe-aproape. Eu, unul, mă raliez celor care spun că nu. Este drept, cheile surogat introduc un anumit grad de redundanță în relații, prin faptul că prezintă, așa cum am văzut în capitolul 7, un grad de artificialitate, adăugându-se celorlalte atribute ce caracterizează „natural" o entitate, un proces, un fenomen. Însă, după cum spunea Date, nu orice redundanță înseamnă denormalizare!

Față de cele declarate în paragraful 7.1 s-ar mai putea spune că, uneori, tentația „surogării" unui grup de atribute (în sensul introducerii unui substitut artificial pentru un atribut/grup de atribute) în vederea simplificării restricțiilor referențiale și a joncțiunilor este irezistibilă. Insist că este vorba de simplificarea joncțiunilor/restricțiilor referențiale, și nu de eliminarea lor. Să luăm un exemplu. De câteva capitole discutăm despre baza de date dedicată gestionării studenților și mai ales a rezultatelor de la examene. În perspectiva procesului „Bologna", învățământul superior va fi organizat pe trei cicluri, licență, specializare+master și doctorat. Așa că structura tabelei STUD ar putea fi:

```
CREATE TABLE stud (
  matricol VARCHAR2(18) PRIMARY KEY,
  numepren VARCHAR2(50),
  cnp CHAR(13),
  ciclu_studii CHAR(1),
  forma_de_studii CHAR(1),
  an_studii NUMBER(1),
  modul_specializare VARCHAR2(30),
  serie_curs NUMBER(1),
  grupa NUMBER(2)
  );
```

Atributul `Ciclu_Studii` ar putea avea doar una dintre valorile 1, 2, 3, `Forma_De_Studii` este *Z* (învățământ de zi) sau *D* (învățământ la distanță), iar `Modul_Specializare` ar avea o dublă folosință: dacă studentul este în primii ani, în care cursurile sunt comune (în aceste semestre, studenții sunt afiliați unor module de studii), atunci valoarea atributului desemnează un modul, în timp ce pentru studenții

înscrişi în ultimele semestre ale ciclului 1 sau în ciclurile 2 sau 3, valoarea semnalează specializarea pe care o urmează. La fiecare modul/specializare, ciclu, formă şi an de studiu pot fi mai multe serii de curs.

În privinţa disciplinelor studiate, n-ar fi prea multe de adăugat:

```
CREATE TABLE discipl (
   coddisc CHAR(6) PRIMARY KEY,
   dendisc VARCHAR2(60),
   nrcreddisc NUMBER(2)
   );
```

Planul de învăţământ se stabileşte la nivel de ciclu/formă/an/modul/specializare:

```
CREATE TABLE curricula (
   ciclu_studii CHAR(1),
   forma_de_studii CHAR(1),
   an_studii NUMBER(1),
   modul_specializare VARCHAR2(30),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   categorie_disc CHAR(1),
   nr_ore_curs NUMBER(3),
   nr_ore_seminar NUMBER(3),
   nr_ore_laborator NUMBER(3),
   PRIMARY KEY (ciclu_studii, forma_de_studii,an_studii,
   modul_specializare, coddisc)
   );
```

unde `Categorie_Disc` indică dacă disciplina respectivă este obligatorie, opţională sau facultativă, iar programarea examenelor se face tot pe cicluri, forme, ani şi module/specializări de studii:

```
CREATE TABLE examene (
   ciclu_studii CHAR(1),
   forma_de_studii CHAR(1),
   an_studii NUMBER(1),
   modul_specializare VARCHAR2(30),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   dataex DATE,
   salaex VARCHAR2(30),
   PRIMARY KEY (ciclu_studii, forma_de_studii,an_studii,
   modul_specializare, coddisc,dataex)
   );
```

Ce-i drept, nu întotdeauna un întreg an de studiu încape într-o sală, dar să presupunem că am avea voie ca valorile atributului `SalaEx` să nu fie scalare.

Nu putem să reproşăm prea multe acestei structuri. Însă, de fiecare dată când vom dori să aflăm ce discipline sunt în planul de învăţământ al formaţiei de studiu din care face parte studentul *Stroici Y. Vasile*, altfel spus, care îi sunt, pentru anul în curs, disciplinele pe care trebuie să le urmeze (obligatorii) sau dintre care trebuie să aleagă (opţionalele), ne-ar trebui o interogare de genul:

```
SELECT dendisc, categorie_disc, nrcreddisc
FROM stud
   INNER JOIN curricula
   ON stud.ciclu_studii = curricula.ciclu_studii
      AND stud.forma_de_studii = curricula.forma_de_studii
      AND stud.an_studii = curricula.an_studii AND
   stud.modul_specializare = curricula.modul_specializare
   INNER JOIN disc
      ON curricula.coddidsc = disc.coddisc
WHERE numepren = 'Stroici Y Vasile'
```

Celor care n-au obosit le putem recomanda călduros să afle în ce zi are programat examenul la *Baze de date* studentul *Pingică I. George*. În asemenea condiţii, să nu-i judecăm prea aspru pe cei care vor apela la un atribut care să substituie faimoasa combinaţie (`Ciclu_Studii`, `Forma_de_Studii`, `An_Studii`, `Modul_Specializare`). Atributul ar fi un surogat în toată regula, pe care l-am putea boteza `IdFormaţie`. Acesta ar fi o sursă nouă de dependenţe funcţionale pentru care constituim o relaţie distinctă. Iată noua structură:

```
CREATE TABLE formaţii (
   idformaţie NUMBER(6) PRIMARY KEY,
   ciclu_studii CHAR(1),
   forma_de_studii CHAR(1)
   an_studii NUMBER(1)
   modul_specializare VARCHAR2(30)
   );
CREATE TABLE stud (
   matricol VARCHAR2(18) PRIMARY KEY,
   numepren VARCHAR2(50),
   cnp CHAR(13),
   idformaţie NUMBER(6) REFERENCES formaţii(idformaţie),
   serie_curs NUMBER(1),
   grupa NUMBER(2)
   );
CREATE TABLE discipl (
   coddisc CHAR(6) NOT NULL PRIMARY KEY,
   dendisc VARCHAR2(60) NOT NULL,
   nrcreddisc NUMBER(2)
   );
```

```
CREATE TABLE curricula (
   idformaţie NUMBER(6) REFERENCES formaţii(idformaţie),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   categorie_disc CHAR(1)
   nr_ore_curs NUMBER(3),
   nr_ore_seminar NUMBER(3),
   nr_ore_laborator NUMBER(3),
   PRIMARY KEY (idformaţie, coddisc)
   );
CREATE TABLE examene (
   idformaţie NUMBER(6) REFERENCES formaţii(idformaţie),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   dataex DATE,
   salaex VARCHAR2(30),
   PRIMARY KEY (idformaţie, coddisc,dataex)
   );
```

Joncţiunile s-au simplificat cu preţul apariţiei unei relaţii noi şi a unei serii de restricţii referenţiale a căror gestiune reclamă timp şi resurse. Rămâne ca dvs. să decideţi dacă merită sau nu efortul...

9.3. Copii „exacte" ale unui atribut în mai multe tabele

Plasarea unui atribut non-cheie într-o tabelă-copil este cazul cel mai frecvent invocat şi, în egală măsură, cel mai puţin recomandat în denormalizare. Să luăm cazul bazei de date pentru evidenţierea notelor studenţilor, discutat ultima dată în paragraful 7.2.3 (*listing*-ul 7.17). Probabil că informaţiile cele mai solicitate din bază se referă la notele obţinute de studenţi:

```
SELECT numepren, studenti.matricol, dendisc, dataex, nota
FROM note INNER JOIN studenti
ON note.matricol=studenti.matricol
   INNER JOIN discipline ON note.coddisc=discipline.coddisc
```

Dacă ne gândim că un asemenea gen de interogare o să se execute de sute de mii sau chiar milioane de ori pe an (la obţinerea a sute de rapoarte, dar şi consultări lansate de studenţi/părinţi folosind o aplicaţie Web conectată la baza de date), ne-ar putea „încolţi" ideea introducerii atributelor `numepren` şi `dendisc` şi în tabela NOTE:

```
ALTER TABLE note ADD (numepren VARCHAR2(50)) ;
ALTER TABLE note ADD (dendisc VARCHAR2(60));
```

Aducerea la zi a valorilor celor două atribute presupune două comenzi `UPDATE` destul de simple:

```
UPDATE note SET numepren =
(SELECT numepren FROM studenti
 WHERE matricol=note.matricol)
```

şi

```
UPDATE note SET dendisc =
(SELECT dendisc FROM discipline
   WHERE coddisc=note.coddisc)
```

Avantajul ar fi că, din moment ce orice notă este furnizată dintr-o singură tabelă, fără a fi nevoie de joncţiuni:

```
SELECT numepren, matricol, dendisc, dataex, nota
FROM note
```

viteza este incomparabil mai mare. Însă nici dezavantajele nu sunt de neglijat. Primul este imensul spaţiu risipit. Dacă luăm în calcul faptul că tabela NOTE poate să numere, la un moment dat, milioane de înregistrări, introducerea celor două atribute costă un spaţiu deloc neglijabil. Cel mai deranjant este însă riscul inconsistenţelor care pot să apară între valorile `NumePren` şi `DenDisc` din NOTE şi cele din tabelele de „origine", STUDENŢI şi DISCIPLINE. De aceea, orice actualizare a celor două atribute în tabelele de origine trebuie să se propage neapărat în NOTE. Lucru posibil doar prin intermediul declanşatoarelor.

Un alt exemplu tentant ar fi cazul practic tratat în paragraful 7.4. Astfel, în tabela DISTRIBUŢIE {`IdFilm`, `Rol`, `Actor`} s-ar economisi timp la obţinerea unei serii de informaţii dacă acestor atribute li s-ar adăuga atributele `TitluOriginal` şi `TitluRo` din FILME.

Fireşte, atributele duplicate pot fi preluate nu numai din tabelele-părinte, ci şi din „bunici", „străbunici" etc. De exemplu, tabela LINII_FACT din baza de date VÂNZĂRI (vezi paragraful 5.2) este cea mai des folosită la reconstituirea valorii facturilor (în lipsa unor atribute calculate `FACTURI.ValFărăTVA`, `FACTURI.ValTotală`). Pentru analiza multidimensională a datelor legate de produsele vândute, putem introduce atât atribute precum data facturii (FACTURI), numele şi cota de TVA ale produsului (PRODUSE), precum şi denumirea clientului (CLIENŢI), localitatea (CODURI_POŞTALE) şi judeţul (JUDEŢE) în care clientul îşi are sediul: LINII_FACT {`NrFact`, `DataFact`, `Linie`, `CodPr`, `DenPr`, `ProcTVA`, `Localitate`, `Judeţ`}. La urma urmei, chiar ne apropiem de OLAP şi depozite de date – o tehnologie care a potenţat denormalizarea.

În această privinţă, mă raliez punctului de vedere potrivit căruia atributele duplicat trebuie evitate aproape întotdeauna! Dacă o aplicaţie merge foarte lent la interogări, trebuie găsite alte explicaţii pentru proasta funcţionare sau, la o adică, trebuie schimbat SGBD-ul! Ideea (schimbării SGBD-ului) nu este atât de neroadă cum pare la prima vedere, astăzi piaţa fiind foarte generoasă la orice categorie de SGBD-uri, inclusiv cele de tip *open source*.

9.4. Atribute derivate/sintetice

Deşi induc un grad de redundanţă semnificativ, atributele derivate sunt privite cu o mult mai mare îngăduinţă. Ba chiar încadrarea lor în tehnicile denormalizării este discutabilă. Pentru prima dată am recurs la atribute derivate în paragraful 7.2.2, în vederea simplificării implementării unor restricţii la nivelul bazei de date. Astfel, pentru baza de date BIBLIOTECĂ, întrucât un titlu (carte) nu poate fi cumpărat în mai mult de 10 exemplare, am convenit că o soluţie elegantă ar fi ca, în tabela TITLURI, să introducem atributul `NrExemplare`, atribut actualizabil exclusiv prin declanşatoarele tabelei EXEMPLARE, restricţia fiind implementabilă printr-o clauză CHECK declarată noului atribut. Iar în cazul bazei de date CONTABILITATE, numărul atributelor derivate introduse în tabela OPERAŢIUNI a fost de 4, utilitatea lor fiind evidentă, ca şi costurile actualizării şi păstrării consistenţei lor. Cele mai importante rezerve cu privire la acest tip de atribute sunt similare celor pomenite la atributele duplicat: volumul suplimentar necesar stocării şi complexitatea mecanismului de menţinere a corectitudinii valorilor acestora.

Dacă unele atribute suplimentare calculate sunt necesare pentru declararea restricţiilor, alteori câştigul urmărit nu ţine decât de creşterea vitezei de acces la o serie de informaţii din bază. Poate cel mai nimerit caz este, în acest sens, cel al bazei de date VÂNZĂRI, adusă în 3NF în finalul paragrafului 5.2, sub forma a opt relaţii: JUDEŢE {`Jud`, `Judeţ`, `Regiune`}, CODURI_POŞTALE {`CodPost`, `Localitate`, `Comună`, `Jud`}, PRODUSE {`CodPr`, `DenPr`, `UM`, `ProcTVA`, `Grupa`}, CLIENŢI2[25] {`CodCl`, `DenCl`, `CodFiscal`, `StradaCl`, `NrStradaCl`, `BlocScApCl`, `TelefonCl`, `CodPost`}, FACT {`NrFact`, `CodCl`, `DataFact`, `Obs`}, LINII_FACT {`NrFact`, `Linie`, `CodPr`, `Cantitate`, `PreţUnit`}, ÎNCASĂRI {`CodÎnc`, `DataÎnc`, `CodDoc`, `NrDoc`, `DataDoc`} şi ÎNCAS_FACT {`CodÎnc`, `NrFact`, `Tranşa`}. Scriptul de creare a acestor tabele poate fi descărcat sub forma *listing*-ului 9.1.

Schema de stocare este una acceptabilă, atâta vreme cât nu luăm în calcul aspectele temporale (vezi subcapitolul următor). Printre cele mai frecvent căutate informaţii din bază sunt şi:

- valoarea unei facturi, fără şi cu TVA;
- valoarea încasată dintr-o factură;
- valoarea rămasă de încasat dintr-o factură (diferenţa dintre valoarea cu TVA a facturii şi valoarea deja încasată).

Determinarea oricăreia dintre cele trei informaţii presupune o serie de joncţiuni, după cum urmează:

```
SELECT SUM(cantitate * pretunit) AS ValoareFaraTVA,
 SUM(cantitate * pretunit * (1 * procTVA)) AS ValTotala,
```

25. Denumim tabela CLIENŢI2, şi nu CLIENŢI, pentru cei care descarcă şi testează *listing*-urile cărţii, astfel încât, la execuţia în aceeaşi schemă, să nu apară o suprapunere cu tabela CLIENŢI de la cazul practic dedicat centrului de închirieri video din capitolul 7.

```
  SUM(NVL(transa,0)) AS ValIncasata,
  SUM(NVL(transa,0)) - SUM(cantitate * pretunit *
  (1 * procTVA)) AS ValRamasaDeÎncasat
FROM fact f
  INNER JOIN linii_fact lf ON f.nrfact=lf.nrfact
  INNER JOIN produse p ON lf.codpr=p.codpr
  LEFT OUTER JOIN incas_fact if ON f.nrfact=if.nrfact
WHERE nrfact = 7545576
```

Fiind un formular cu regim special, din momentul confirmării, o factură nu poate suferi modificări. La fel stau lucrurile şi cu ordinele de plată, extrasele de cont şi alte documente ce atestă încasarea facturilor. Prin urmare, în timp, declanşatoarele de modificare şi de ştergere din FACT, LINII_FACT şi ÎNCAS_FACT vor fi extrem de rar executate.

Pe acest considerent, ne putem gândi să introducem în tabela FACT trei atribute, `ValTotală`, `TVAFact` şi `ValÎncasată`. Toate trei vor fi calculate automat prin declanşatoarele de inserare, modificare şi ştergere a tabelelor LINII_FACT şi ÎNCAS_FACT:

```
ALTER TABLE fact ADD ValTotală NUMBER (15) ;
ALTER TABLE fact ADD TVAFact NUMBER (15) ;
ALTER TABLE fact ADD ValÎncasată NUMBER (15) ;
```

Cele două declanşatoare de inserare ar trebui să funcţioneze după următoarea logică, prezentată în *listing*-ul 9.2:

Listing 9.2. Declanşatoarele de inserare în LINII_FACT şi ÎNCAS_FACT

```
CREATE OR REPLACE TRIGGER trg_lf_ins
   AFTER INSERT ON linii_fact FOR EACH ROW
BEGIN
   UPDATE fact SET
      valtotala = NVL(valtotala,0) + :NEW.cantitate *:NEW.pretunit * (1 +
         (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
      TVAfact = NVL(TVAfact,0) + :NEW.cantitate * :NEW.pretunit *
         (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)
      WHERE nrfact=:NEW.nrfact ;
END ;
/
CREATE OR REPLACE TRIGGER trg_lf_ins
   AFTER INSERT ON incas_fact FOR EACH ROW
BEGIN
   UPDATE fact
   SET ValIncasata = NVL(ValIncasata,0) + :NEW.transa
   WHERE nrfact=:NEW.nrfact ;
END ;
/
```

Declanşatoarele de modificare şi ştergere reclamă efort doar de scriere şi implementare, întrucât execuţia lor de către SGBD este foarte rară. Importanţa lor este însă esenţială în menţinerea corectitudinii valorilor pentru atributele calculate/sintetice – vezi *listing*-ul 9.3.

Listing 9.3. Declanşatoarele de modificare şi ştergere ale tabelelor LINII_FACT şi ÎNCAS_FACT

```
-- actualizare in LINII_FACT
CREATE OR REPLACE TRIGGER trg_lf_upd
    AFTER UPDATE ON linii_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vechea factura
        UPDATE fact SET                -- si adaugat la noua
            valtotala = valtotala – :OLD.cantitate *:OLD.pretunit * (1 +
                (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)),
            TVAfact = TVAfact – :OLD.cantitate * :OLD.pretunit *
                (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
        WHERE nrfact=:NEW.nrfact ;

        UPDATE fact SET
            valtotala = valtotala + :NEW.cantitate *:NEW.pretunit * (1 +
                (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
            TVAfact = TVAfact + :NEW.cantitate * :NEW.pretunit *
                (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)
        WHERE nrfact=:NEW.nrfact ;

    ELSE  -- nu s-a schimbat numarul facturii

        UPDATE fact SET
            valtotala = valtotala – :OLD.cantitate *:OLD.pretunit * (1 +
            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)) +
            :NEW.cantitate *:NEW.pretunit * (1 +
                (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
               TVAfact = TVAfact – :OLD.cantitate * :OLD.pretunit *
                (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
        WHERE nrfact=:NEW.nrfact ;
    END IF ;
END ;
/

-- stergere in LINII_FACT
CREATE OR REPLACE TRIGGER trg_lf_del
    BEFORE DELETE ON linii_fact FOR EACH ROW
BEGIN
    UPDATE fact SET
        valtotala = valtotala – :OLD.cantitate *:OLD.pretunit * (1 +
            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)),
        TVAfact = TVAfact – :OLD.cantitate * :OLD.pretunit *
            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
```

```
    WHERE nrfact=:OLD.nrfact ;
END ;
/

-- actualizare in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_upd
    AFTER UPDATE ON incas_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vechela factura
        UPDATE fact SET                -- si adaugat la noua
            ValIncasata = ValIncasata – :OLD.transa
        WHERE nrfact=:OLD.nrfact ;

        UPDATE fact SET ValIncasata = ValIncasata + :NEW.transa
        WHERE nrfact=:NEW.nrfact ;

    ELSE
        UPDATE fact SET ValIncasata = ValIncasata – :OLD.transa – :NEW.transa
        WHERE nrfact=:NEW.nrfact ;
    END IF;
END ;
/

-- stergere in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_del
    BEFORE DELETE ON incas_fact FOR EACH ROW
BEGIN
    UPDATE fact
    SET ValIncasata = ValIncasata – :OLD.transa
    WHERE nrfact=:OLD.nrfact ;
END ;
/
```

Atenţie, însă! Actualizarea atributelor sintetice pe baza declanşatoarelor este necesară, dar nu şi suficientă în asigurarea corectitudinii lor. Ce s-ar întâmpla dacă, după câtva timp de funcţionare a aplicaţiei, cineva (nu spunem cine!) ar „scăpa" o comandă de genul:

```
UPDATE fact
SET ValTotală = 555555 ;
```

Toate liniile tabelei FACT ar fi afectate, iar valoarea totală a fiecărei facturi ar fi cu totul aiurea. Oficial, pentru cei neiniţiaţi, s-ar putea lansa ideea atacului unui virus, dar soluţia aceasta are o viaţă scurtă. Trebuie găsit un mecanism prin care atributele calculate să fie modificate numai prin declanşatoare. În Oracle, o soluţie nu prea complicată ţine de folosirea unor variabile publice care să semnaleze că modificarea se face din declanşatorul autorizat. Iată, în *listing*-ul 9.4, o asemenea variantă de lucru.

Listing 9.4. Pachetul, noile declanşatoare ale tabelelor LINII_FACT şi ÎNCAS_FACT, plus declanşatoarele tabelei FACT pentru controlul actualizării atributelor sintetice

```
CREATE OR REPLACE PACKAGE pac_vanzari AS
    v_trg_liniifact BOOLEAN := FALSE ;
    v_trg_incas_fact BOOLEAN := FALSE ;
END pac_vanzari ;
/
-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_ins
    AFTER INSERT ON linii_fact FOR EACH ROW
BEGIN
    pac_vanzari.v_trg_liniifact := TRUE ;
    UPDATE .... – comanda este identică celei din listing-ul 9.3
    pac_vanzari.v_trg_liniifact := FALSE ;
END ;
/
-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_ins
     AFTER INSERT ON incas_fact FOR EACH ROW
BEGIN
    pac_vanzari.v_trg_incas_fact := TRUE ;
    UPDATE .... – comanda este identică celei din listing-ul 9.3
    pac_vanzari.v_trg_incas_fact := FALSE ;
END ;
/
-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_upd
    AFTER UPDATE ON linii_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vechea factura
        pac_vanzari.v_trg_liniifact := TRUE ;
        UPDATE .... – comanda este identică celei din listing-ul 9.3
        UPDATE .... – comanda este identică celei din listing-ul 9.3
        pac_vanzari.v_trg_liniifact := FALSE ;
    ELSE  -- nu s-a schimbat numarul facturii
        pac_vanzari.v_trg_liniifact := TRUE ;
        UPDATE .... – comanda este identică celei din listing-ul 9.3
        pac_vanzari.v_trg_liniifact := FALSE ;
    END IF ;
END ;
/
-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_del
    BEFORE DELETE ON linii_fact FOR EACH ROW
BEGIN
    pac_vanzari.v_trg_liniifact := TRUE ;
    UPDATE .... – comanda este identică celei din listing-ul 9.3
    pac_vanzari.v_trg_liniifact := FALSE ;
END ;
/
-----------------------------------------------------------------------
```

```
-- actualizare in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_upd
    AFTER UPDATE ON Incas_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vechea factura
        pac_vanzari.v_trg_incas_fact := TRUE ;
        UPDATE .... – comanda este identică celei din listing-ul 9.3
        pac_vanzari.v_trg_incas_fact := FALSE ;
    ELSE
        pac_vanzari.v_trg_incas_fact := TRUE ;
        UPDATE .... – comanda este identică celei din listing-ul 9.3
        pac_vanzari.v_trg_incas_fact := FALSE ;
    END IF;
END ;
/
-------------------------------------------------------------------------------
-- stergere in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_del
    BEFORE DELETE ON Incas_fact FOR EACH ROW
BEGIN
    pac_vanzari.v_trg_incas_fact := TRUE ;
    UPDATE .... – comanda este identică celei din listing-ul 9.3
    pac_vanzari.v_trg_incas_fact := FALSE ;
END ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_ins
    BEFORE INSERT ON fact FOR EACH ROW
BEGIN
    :NEW.ValTotala := 0 ;
    :NEW.TVAFact  := 0 ;
    :NEW.ValIncasata := 0 ;
END ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_upd2
    AFTER UPDATE OF ValTotala, TVAFact, ValIncasata
    ON fact FOR EACH ROW
BEGIN
    IF :NEW.ValTotala <> :OLD.ValTotala OR :NEW.TVAFact <> :OLD.TVAFact THEN
        IF pac_vanzari.v_trg_liniifact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876,
            'ValTotala si TVAFact nu pot fi modificate interactiv!');
        END IF ;
    END IF ;

    IF :NEW.ValIncasata <> :OLD.ValIncasata THEN
        IF pac_vanzari.v_trg_incas_fact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876,
            'ValIncasata nu poate fi modificata interactiv!');
```

```
      END IF ;
   END IF ;
END ;
/
```

Pentru un plus de lizibilitate, elementele de noutate faţă de *listing*-ul 9.3 au fost puse în evidenţă prin îngroşare. *Listing*-ul descărcabil de pe Web este însă complet.

Urmând această idee, ne mai putem gândi la câteva atribute consultate în mod frecvent, ce ar putea fi introduse în schemă şi calculate prin declanşatoare, în condiţiile în care calculul lor ar implica joncţiuni:

- în tabela PRODUSE, s-ar putea introduce atributul `Vânzări`, incrementabil la fiecare prezenţă a produsului respectiv pe linia vreunei facturi; actualizarea sa este operată din cele trei declanşatoare ale LINII_FACT;
- în tabela ÎNCASĂRI, atributul `ValoareÎncasare`, actualizabil prin declanşatoarele tabelei ÎNCAS_FACT;
- în tabela CLIENŢI:
  - `Vânzări` ar cumula valorile facturilor emise destinate clientului respectiv, fiind actualizabil prin declanşatoarele tabelei FACT (în condiţiile folosirii `ValTotală, TVAFact`);
  - `Plăţi` ar însuma toate încasările de la client, corectitudinea valorii sale fiind asigurată prin intermediul celor trei declanşatoare ale tabelei FACT, de data aceasta atributul împricinat fiind `ValÎncasată`.

Interesant este că o serie de asemenea atribute ar putea fi utile şi pentru implementarea unor reguli mai sofisticate, precum cele discutate în paragraful 7.2.2. Astfel, firma pentru care este proiectată schema bazei de date VÂNZĂRI decide să nu mai vândă nimic nici unui client care are datorii mai mari de 200 de milioane de lei sau clienţilor la care datoriile sunt mai mari de 30% din valoarea vânzărilor. Cu aportul ultimelor două atribute introduse în CLIENŢI, această restricţie devine o banalitate:

```
ALTER TABLE clienţi ADD CHECK
(vânzări - plăţi <= 200000000) ) ;
```

în cazul primei formulări sau:

```
ALTER TABLE clienţi ADD CHECK
((vânzări - plăţi) / vânzări <= .3 ) ;
```

pentru a o onora pe cea de-a doua.

În concluzie, deşi nu suntem forţaţi să o facem, am putea delimita două categorii de atribute redundante derivate/sintetice: cea din paragraful 7.2.2, care grupează atributele necesare implementării unor restricţii mai complexe, şi categoria atributelor

utile în obţinerea unor informaţii (punctuale sau rapoarte în toată regula) din bază cu un consum de resurse cât mai mic sau, în fine, cât mai judicios.

Gradul lor de utilitate trebuie raportat la costurile integrităţii şi coerenţei valorilor lor, uneori meritând efortul, alteori nu. Un caz simplu pe care-l putem încadra în categoria celor care merită efortul este cel al tabelei LINII_FACT, în care introducem un atribut de genul `ValLinieCuTVA`, adică valoarea cu TVA a produsului de pe linia curentă a facturii, calculată prin expresia: `LINII_FACT.Cantitate * LINIIFACT.PreţUnit * (1 + PRODUSE.ProcTVA)`. Argumentul principal ţine de regimul facturilor, care, după momentul introducerii, nu mai pot fi modificate, în schimb, numărul consultărilor ulterioare în care sunt implicate înregistrările respective este imens.

Vom vedea în subcapitolul următor că, prin ricoşeu, acest atribut redundant ne face un mare serviciu în materie de valabilitate în timp a schemei bazei, în condiţiile în care, la câţiva ani, procentul de TVA al unui produs se poate modifica. Printre alte atribute relativ utile se numără cele trei propuse anterior în acest paragraf, `FACT.ValTotală`, `FACT.TVAFact`, `FACT.ValÎncasată`, `CLIENŢI.Vânzări`, `CLIENŢI.Plăţi`.

Chiar dacă utile, majoritatea acestor atribute au o doză artificială şi sunt tratate, uneori, cu nemeritat dispreţ. Am vrea să vă propunem şi o speţă de redundanţă care este pe cât de exagerată, pe atât de utilă. După repetate încercări, baza de date CONTABILITATE a fost adusă la o schemă rezonabilă, chiar dacă cu destule atribute redundante. Graful final al dependenţelor este cel din figura 7.10. Ei bine, pentru a obţine şi cele mai simple documente contabile (nu ştim care sunt acelea) ne lipseşte o informaţie esenţială: soldul iniţial a conturilor. Mulţumim pe această cale celor care se pricep la contabilitate şi au păstrat discreţia până în acest punct al cărţii. Trebuie degrabă introdus aşadar atributul `SoldIniţial`, care depinde funcţional de atributul `ContElementar`. Necazul e că soldul iniţial al unui cont poate fi debitor sau creditor.

Pentru conturile de activ, soldurile iniţiale sunt debitoare, iar pentru cele de pasiv, soldurile iniţiale sunt creditoare. În schimb, pentru o serie de conturi bifuncţionale, soldul iniţial poate fi debitor, în timp ce alte conturi bifuncţionale prezintă sold iniţial creditor. Drept pentru care, deşi pare o crasă redundanţă, introducem nu un atribut pentru soldul iniţial, ci două: sold iniţial debitor (`SoldInitBD`) şi sold iniţial creditor (`SoldInitCR`). Ba, mai mult, pentru a avea în orice moment „la îndemână" soldul curent al contului, vom recurge la două atribute denumite pompos calculate/sintetizatoare/centralizatoare: rulaj debitor (`RulajDB`) şi rulaj creditor (`RulajCR`). Forma uşor schimbată a grafului de dependenţe funcţionale/de incluziune este cea din figura 9.1.

Noile atribute afectează doar tabela CONTURI_ELEMENTARE care, mai nou, are structura: CONTURI_ELEMENTARE {`ContElementar`, `SoldInitDB`, `SoldInitCR`, `RulajDB`, `RulajCR`}.

```
ALTER TABLE conturi_elementare ADD SoldInitDB NUMBER(14) ;
ALTER TABLE conturi_elementare ADD SoldInitCR NUMBER(14) ;
ALTER TABLE conturi_elementare ADD RulajDB NUMBER(14) ;
ALTER TABLE conturi_elementare ADD RulajCR NUMBER(14) ;
```

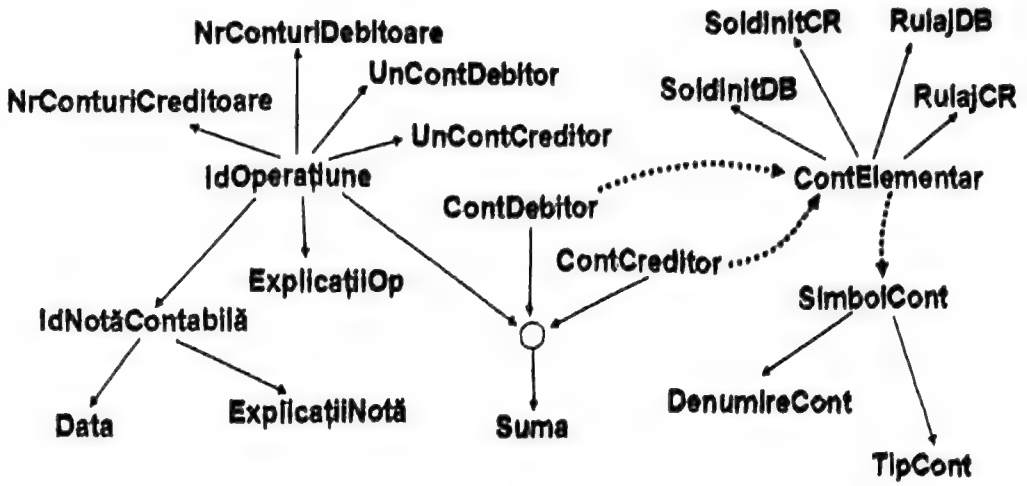


Figura 9.1. Noi atribute, care mai de care mai redundante

Un cont, oricare ar fi natura sa, nu poate avea simultan sold iniţial şi pe debit, şi pe credit, aşa încât avem nevoie de o restricţie de genul:

```
ALTER TABLE conturi_elementare
ADD CHECK (NVL(SoldInitDB,0) * NVL(SoldInitCR,0) = 0) ;
```

Un alt exemplu în care folosirea unui atribut derivat/calculat este crucială priveşte o aplicaţie prin care o bancă gestionează conturile de card folosite la bancomate sau pentru plăţi în supermagazine. Contul din care se pot face plăţi sau retrage numerar cu ajutorul cardului este identificat prin `IdCont`, celelalte informaţii despre cont fiind: numărul (`NrCont`), tipul (`TipCont`), data la care a fost deschis (`DataDeschiderii`), soldul la deschidere (`SoldIniţial`) şi titularul (`CNPPosesor`). O dată deschis un asemenea cont, există două mari categorii de operaţiuni (`IdOperaţiune`, `DataOraOp`) cu acesta: *alimentări* (virarea unor sume în acest cont, operaţiune care se numeşte, în termeni tehnici, alimentare a contului – `IdAlimCont`, `SumaAlimCont`) şi *plăţi* (`IdPlată`, `TipPlată`, `SumăPlată`, `ProcentComision`). În funcţie de tipul plăţii, banca percepe un comision exprimat ca procent aplicat la suma plătită.

Categoria *plăţi* cuprinde, la rândul său, trei tipuri de operaţiuni:

- *retragere de numerar de la bancomat* (prilej, uneori, de cozi similare celor de la alimentarele de acum mai bine de 15 ani). Fiecare retragere de acest tip are un identificator (`IdRetrNumerar`), iar a doua informaţie vitală pentru consemnarea sa este de la ce bancomat s-a făcut retragerea numerarului, fiecare bancomat fiind caracterizat printr-un identificator (`IdBancomat`), un număr „oficial" (`NrBcmt`) ce i-a fost alocat de banca de care aparţine (`Bancă`) şi adresa la care este situat (`AdresaBcmt`);
- *plăţi ale unor facturi direct de la bancomat* (de exemplu, facturile Orange sau Connex), identificate printr-un atribut special (`IdPlatăBancomat`), plăţi care, spre deosebire de retragerile de numerar, presupun virarea banilor într-un cont destinaţie (`IdContDest`) ce aparţine beneficiarului plăţii. Despre contul destinaţie, în afara identificatorului, mai interesează: numărul său (`NrContDest`) şi mai recentul identificator internaţional IBAN (`IBAN_ContDest`), precum şi banca la care a fost deschis;

- *plăţi efectuate de la puncte de vânzare* (*Point of Sale*) amplasate frecvent în supermagazine, staţii de benzină etc., prevăzute cu un dispozitiv special în care se introduce cardul, se tastează codul PIN şi se face transferul banilor. Acest tip de plată este identificat, fireşte, printr-un atribut special (`IdPlPunctVânz`), iar elementele care interesează în mod deosebit privesc ID-ul punctului de vânzare (`IdPunctVânzare`), denumirea (`DenPV`) şi adresa sa (`AdresaPV`), precum şi contul în care are loc virarea banilor (`IdContDest`).

După sistematizare, graful dependenţelor funcţionale poate arăta precum cel din figura 9.2.

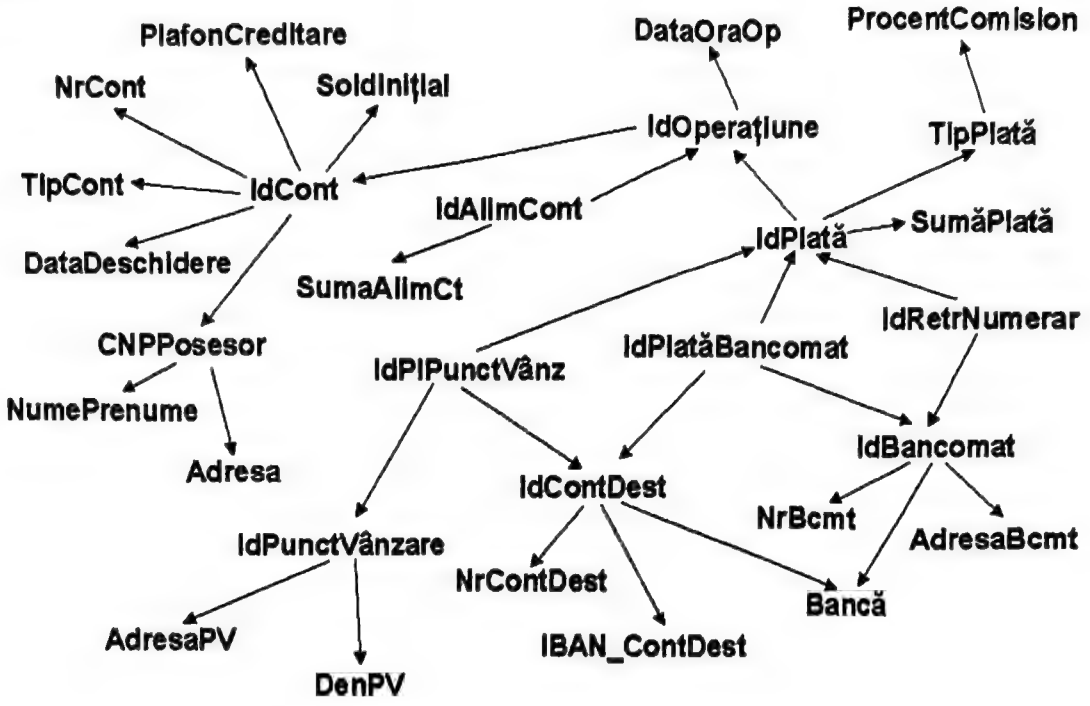


Figura 9.2. Graful DF pentru baza de date BANCOMATE

Relaţiile decupate pe baza acestui graf, precum şi câteva dintre restricţiile ce pot fi instituite sunt prezentate în scriptul Oracle din *listing*-ul 9.5.

Listing 9.5. Scriptul de creare a tabelelor bazei de date BANCOMATE

```
DROP TABLE plati_puncte_vanzare ;
DROP TABLE puncte_vanzare ;
DROP TABLE plati_prin_bancomat ;
DROP TABLE conturi_destinatie ;
DROP TABLE retrageri_numerar ;
DROP TABLE bancomate ;
DROP TABLE plati ;
DROP TABLE tipuri_plati  ;
DROP TABLE alimentare_conturi ;
DROP TABLE operatiuni_card ;
DROP TABLE conturi_card ;
DROP TABLE titulari_card ;
```

```
CREATE TABLE titulari_card (
    CNPPosesor CHAR(13) NOT NULL PRIMARY KEY,
    NumePrenume VARCHAR2(45) NOT NULL,
    Adresa VARCHAR2(60)
);

CREATE TABLE conturi_card (
    IdCont NUMBER(14) NOT NULL PRIMARY KEY,
    CNPPosesor CHAR(13) NOT NULL REFERENCES titulari_card (CNPPosesor),
    DateDeschidere DATE DEFAULT CURRENT_DATE NOT NULL,
    TipCont VARCHAR2(15) NOT NULL,
    NrCont NUMBER(16) NOT NULL,
    PlafonCreditare NUMBER(14) NOT NULL,
    SoldInitial NUMBER(14) NOT NULL
);

CREATE TABLE operatiuni_card (
    IdOperatiune NUMBER(16) NOT NULL PRIMARY KEY,
    DataOraOp DATE DEFAULT CURRENT_DATE NOT NULL,
    IdCont NUMBER(14) NOT NULL REFERENCES conturi_card (IdCont)
);

CREATE TABLE alimentare_conturi (
    IdAlimCont NUMBER(15) NOT NULL PRIMARY KEY,
    SumaAlimCont NUMBER (14) NOT NULL CHECK (SumaAlimCont > 0),
    IdOperatiune NUMBER(16) NOT NULL REFERENCES operatiuni_card (IdOperatiune)
);

CREATE TABLE tipuri_plati (
    TipPlata VARCHAR2(15) NOT NULL PRIMARY KEY,
    ProcentComision NUMBER(4,4)
);

CREATE TABLE plati (
    IdPlata NUMBER(15) NOT NULL PRIMARY KEY,
    TipPlata VARCHAR2(15) NOT NULL REFERENCES tipuri_plati (TipPlata),
    SumaPlata NUMBER(14),
    IdOperatiune NUMBER(20) NOT NULL REFERENCES operatiuni_card (IdOperatiune)
);

CREATE TABLE bancomate (
    IdBancomat NUMBER(7) NOT NULL PRIMARY KEY,
    Banca VARCHAR2(30) NOT NULL,
    NrBcmt NUMBER(5) NOT NULL,
    AdresaBcmt VARCHAR2(50)
);

CREATE TABLE retrageri_numerar (
    IdRetrNumerar NUMBER(15) NOT NULL PRIMARY KEY,
    IdPlata NUMBER(15) NOT NULL REFERENCES plati (IdPlata),
    IdBancomat NUMBER(7) NOT NULL REFERENCES bancomate (IdBancomat)
);
```

```
CREATE TABLE conturi_destinatie (
    IdContDest NUMBER(10) NOT NULL PRIMARY KEY,
    NrContDest NUMBER(20) NOT NULL,
    Banca VARCHAR2(30) NOT NULL,
    IBAN_ContDest VARCHAR2(20) NOT NULL
);

CREATE TABLE plati_prin_bancomat (
    IdPlataBancomat NUMBER(15) NOT NULL PRIMARY KEY,
    IdPlata NUMBER(15) NOT NULL REFERENCES plati (IdPlata),
    IdBancomat NUMBER(7) NOT NULL REFERENCES bancomate (IdBancomat),
    IdContDest NUMBER(10) NOT NULL REFERENCES conturi_destinatie (IdContDest)
);

CREATE TABLE puncte_vanzare (
    IdPunctVanzare NUMBER(10) NOT NULL PRIMARY KEY,
    DenPV VARCHAR2(30),
    AdresaPV VARCHAR2(50)
);

CREATE TABLE plati_puncte_vanzare (
    IdPlPunctVanz NUMBER(14) NOT NULL PRIMARY KEY,
    IdPlata NUMBER(15) NOT NULL REFERENCES plati (IdPlata),
    IdPunctVanz NUMBER(10) NOT NULL REFERENCES puncte_vanzare (IdPunctVinzare),
    IdContDest NUMBER(10) NOT NULL REFERENCES conturi_destinatie (IdContDest)
);
```

Ei bine, una dintre cele mai importante reguli ale aplicaţiei este: în urma alimentărilor şi plăţilor, un cont nu poate avea un deficit mai mare decât plafonul de creditare. Spre exemplu, un client deschide un cont pentru card în care depune 10 milioane de lei (`SoldIniţial`) şi stabileşte cu banca un plafon de creditare de 5 milioane de lei. După o primă retragere de numerar de 7 milioane de lei, soldul contului rămâne de 3 milioane. Totuşi, după o alimentare cu două milioane, alimentare în urma căreia soldul ajunge la 5 milioane, clientul poate plăti/retrage din cont 10 milioane de lei. Soldul său este de minus 5 milioane de lei, iar din acest moment, până la o nouă alimentare a contului, nu mai poate face nici o plată/retragere.

La fiecare plată, aplicaţia trebuie, deci, neapărat să verifice dacă suma ce ar urma a fi plătită nu depăşeşte soldul curent plus plafonul de creditare. Or, verificarea presupune următorul calcul: soldul iniţial + toate alimentările – toate plăţile + plafonul de creditare > suma ce se doreşte a fi plătită. Dacă banca are sute de mii de asemenea conturi, în câţiva ani numărul operaţiunilor se ridică la milioane sau chiar zeci-sute de milioane, iar verificarea de mai sus devine extrem de consumatoare de resurse. Situaţia impune recurgerea la un nou atribut, `SoldCurent`, dependent funcţional de `IdCont`, care va fi adăugat tabelei CONTURI_CARD:

```
ALTER TABLE conturi_card ADD (SoldCurent NUMBER(14)) ;
```

Gestionarea sa se va face similar atributelor sintetice din schema bazei de date VÂNZĂRI, prin mecanismul în doi timpi, actualizarea pe baza declanşatoarelor

altor tabele, plus protejarea la modificările interactive folosind declanşatorul de modificare al tabelei CONTURI_CARD.

Schimbând un pic registrul, facem apel la neuitatul capitol 7, mai precis la paragraful dedicat centrului de închirieri de casete video. În *listing*-ul 7.29 ne-am încordat forţele pentru a gestiona automat, prin declanşatoare, valoarea atributului `FilmNr` din tabela CASETE_FILME. Acest atribut trebuia să se incrementeze automat la adăugarea unui film pe o casetă. Asemănarea cu atributul `Linie` din tabela LINII_FACT de la începutul acestui paragraf, cu atributul `OrdineCopertă` din tabela CĂRŢI_AUTORI (vezi figura 8.27 din paragraful 8.4) sau cu `LocalitNr` din LOC_RUTE (vezi figura 8.23 din paragraful 8.3.3) este absolut tulburătoare. Dificultatea acestor atribute ţine de faptul că valorile lor trebuie să fie ca ale unui număr curent într-un tabel obişnuit, iar ştergerea unei linii din tabelele menţionate impune reaşezarea valorilor acestor numere curente.

Lucrurile s-ar putea simplifica într-o oarecare măsură dacă în tabelele-părinte CASETE, FACT, CĂRŢI şi RUTE am adăuga un atribut, firește, sintetic, care să conţină numărul liniilor din respectivele casete, facturi, cărţi şi rute. Să continuăm cu tandemul FACT - LINII_FACT din acest paragraf. Mai întâi, adăugăm atributul `NrLinii` în tabela FACT:

```
ALTER TABLE fact ADD (NrLinii NUMBER(2)) ;
```

Rescriem pachetul PAC_VÂNZĂRI, adăugând o funcţie care primeşte drept parametru numărul facturii şi returnează valoarea atributului `NrLinii`. Apoi, în declanşatorul de inserare pentru FACT, avem grijă ca valoarea lui `NrLinii` de pe linia nou introdusă să fie zero, iar în cel de modificare se verifică dacă eventuala modificare a acestui atribut provine din declanşatoarele tabelei LINII_FACT, semnalizarea fiind posibilă prin variabila publică V_TRG_LINIIFACT - vezi *listing*-ul 9.6, partea I.

Listing 9.6 (partea I). Pachetul, declanşatoarele actualizate de inserare şi modificare ale tabelei FACT

```
-- ==== Pachetul PAC_VANZARI
-----------------------------------------
CREATE OR REPLACE PACKAGE pac_vanzari AS
    v_trg_liniifact BOOLEAN := FALSE ;
    v_trg_incas_fact BOOLEAN := FALSE ;
    FUNCTION f_nrlinii (nrfact_ fact.NrFact%TYPE)
        RETURN fact.NrLinii%TYPE ;
    v_nrfact fact.NrFact%TYPE ;
    v_liniestearsa linii_fact.Linie%TYPE ;
    v_trg_liniifact_del BOOLEAN := FALSE ;
END pac_vanzari ;
/
-----------------------------------------
```

```
CREATE OR REPLACE PACKAGE BODY pac_vanzari AS
    FUNCTION f_nrlinii (nrfact_ fact.NrFact%TYPE)  RETURN fact.NrLinii%TYPE
IS
    v_nrlinii fact.NrLinii%TYPE ;
BEGIN
    SELECT NrLinii INTO v_nrlinii FROM fact WHERE NrFact = nrfact_ ;
    RETURN v_nrlinii ;
END f_nrlinii ;
END pac_vanzari ;
/

-- ==== Declansatoarele de inserare si modificare pentru FACT
-------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_ins
    BEFORE INSERT ON fact FOR EACH ROW
BEGIN
    :NEW.ValTotala := 0 ;
    :NEW.TVAFact  := 0 ;
    :NEW.ValIncasata := 0 ;
    :NEW.NrLinii := 0 ;
END ;
/
-------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_upd2
    AFTER UPDATE OF ValTotala, TVAFact,  ValIncasata, NrLinii
    ON fact FOR EACH ROW
BEGIN
    -- atribute modificate din LINII_FACT
    IF :NEW.ValTotala <> :OLD.ValTotala OR :NEW.TVAFact <> :OLD.TVAFact
        OR :NEW.NrLinii <> :OLD.NrLinii THEN
        IF pac_vanzari.v_trg_liniifact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876,
            'NrLinii, ValTotala si TVAFact nu pot fi modificate interactiv!');
        END IF ;
    END IF ;
    -- atribute modificate din INCAS_FACT
    IF :NEW.ValIncasata <> :OLD.ValIncasata THEN
        IF pac_vanzari.v_trg_incas_fact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876, 'ValIncasata nu poate fi modificata
            interactiv!');
        END IF ;
    END IF ;
END ;
/
```

De acum începe partea ceva mai interesantă. Păstrând declanşatorul de inserare la nivel de linie, de tip AFTER (TRG_LF_INS), se creează, tot pentru inserare, un altul – TRG_LF_INS0 – de tip BEFORE-ROW, dedicat incrementării valorii atributului `NrLinii` din FACT – vezi *listing*-ul 9.6, partea a II-a. Mecanismul de gestionare a ştergerilor din LINII_FACT şi actualizare corespunzătoare a atributelor

sintetice din FACT este alcătuit din două declanşatoare de ştergere, unul la nivel de linie (BEFORE-ROW) şi altul la nivel de comandă (AFTER-STATEMENT), cu numele TRG_LF_DEL, respectiv TRG_LF_DEL2.

Listing 9.6 (partea a II-a). Un nou declanşator de inserare şi două de ştergere pentru tabela LINII_FACT

```
-- == Declansatoarele tabelei LINII_FACT
-------------------------------------------
-- se pastreaza TRG_LF_INS

-------------------------------------------
-- nou declansator de inserare (BEFORE-ROW)
CREATE OR REPLACE TRIGGER trg_lf_ins0 BEFORE INSERT ON linii_fact FOR EACH
ROW
BEGIN
    pac_vanzari.v_trg_liniifact := TRUE ;
    :NEW.Linie := pac_vanzari.f_nrlinii (:NEW.NrFact) + 1 ;
    UPDATE fact SET NrLinii = NrLinii + 1 WHERE NrFact = :NEW.NrFact ;
    pac_vanzari.v_trg_liniifact := FALSE ;
END ;
/
-------------------------------------------
-- stergere in LINII_FACT – la nivel de LINIE
CREATE OR REPLACE TRIGGER trg_lf_del
    BEFORE DELETE ON linii_fact FOR EACH ROW
BEGIN
    pac_vanzari.v_trg_liniifact := TRUE ;
    pac_vanzari.v_nrfact := :OLD.NrFact ;
    pac_vanzari.v_liniestearsa := :OLD.Linie ;
    UPDATE fact SET
        valtotala = valtotala – :OLD.cantitate *
            :OLD.pretunit * (1 +
             (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)),
        TVAfact = TVAfact – :OLD.cantitate * :OLD.pretunit *
            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
    WHERE nrfact=:OLD.nrfact ;
    pac_vanzari.v_trg_liniifact := FALSE ;
END ;
/
-------------------------------------------
-- stergere in LINII_FACT – la nivel de COMANDA
CREATE OR REPLACE TRIGGER trg_lf_del2
    AFTER DELETE ON linii_fact
BEGIN
    IF pac_vanzari.v_nrfact IS NULL OR pac_vanzari.v_liniestearsa IS NULL THEN
        -- nu s-a sters nici o linie
        NULL ;
    ELSE
        pac_vanzari.v_trg_liniifact := TRUE ;
```

```
        IF pac_vanzari.f_nrlinii(pac_vanzari.v_nrfact) =
              pac_vanzari.v_liniestearsa THEN
           -- linia stearsa din factura este ultima
           NULL ;
        ELSE
           -- liniei sterse i se da un numar mai mare pentru a nu se viola cheia primara
           pac_vanzari.v_trg_liniifact_del := TRUE ;
           UPDATE linii_fact
           SET linie =  pac_vanzari.f_nrlinii(pac_vanzari.v_nrfact) + 1
           WHERE NrFact = pac_vanzari.v_nrfact AND linie = pac_vanzari.v_liniestearsa ;

           -- se scade valoarea atributului LINIE pentru liniile ulterioare celei sterse
           UPDATE linii_fact
           SET linie =  linie – 1
           WHERE NrFact = pac_vanzari.v_nrfact AND linie > pac_vanzari.v_liniestearsa ;
           pac_vanzari.v_trg_liniifact_del := FALSE ;
        END IF ;
        UPDATE fact
        SET NrLinii =  NrLinii – 1
        WHERE NrFact = pac_vanzari.v_nrfact ;

        pac_vanzari.v_nrfact := NULL ;
        pac_vanzari.v_liniestearsa := NULL ;
        pac_vanzari.v_trg_liniifact := FALSE ;
     END IF ;
END ;
/
```

Declanşatorul de ştergere la nivel de linie stochează numărul facturii şi pe cel al liniei în două variabile publice, iar declanşatorul la nivel de comandă renumerotează liniile în LINII_FACT şi actualizează atributele calculate în FACT. Atenţie! Mecanismul este gândit pentru ştergerea unei singure linii la o comandă DELETE; dacă se introduce un DELETE mai generos, de genul: `DELETE FROM linii_fact WHERE NrFact >= 3`, lucrurile se încâlcesc rău de tot!

Ultima parte a *listing*-ului 9.6 prezintă cele două declanşatoare de modificare pentru tabela LINII_FACT. Sunt necesare două, la nivel de linie (BEFORE-ROW) şi la nivel de comandă (AFTER-STATEMENT), întrucât ţinem morţiş ca valorile atributului linie să fie corecte, deci recorectate după fiecare modificare a `NrFact`.

Listing 9.6 (partea a III-a). Declanşatoarele de modificare pentru LINII_FACT

```
-----------------------------------------
-- declansatorul de actualizare in LINII_FACT de tip BEFORE-ROW
CREATE OR REPLACE TRIGGER trg_lf_upd
   BEFORE UPDATE ON linii_fact FOR EACH ROW
BEGIN
   IF :NEW.Linie <> :OLD.Linie AND pac_vanzari.v_trg_liniifact = FALSE THEN
      RAISE_APPLICATION_ERROR (-20870, 'Atributul Linie nu poate fi modificat
      direct!');
   END IF ;
```

```
    IF pac_vanzari.v_trg_linilfact_del = FALSE THEN
        pac_vanzari.v_trg_linilfact := TRUE ;
        IF :NEW.nrfact <> :OLD.nrfact THEN
            -- linia a fost trasferata in alta factura
            --       mai intii, se scade valoarea, TVA-ul, si se renumeroteaza
            --           liniile facturii din care "a plecat"
            pac_vanzari.v_nrfact := :OLD.NrFact ;
            pac_vanzari.v_liniestearsa := :OLD.Linie ;
            UPDATE fact SET
                NrLinii = NrLinii – 1,
                valtotala = valtotala – :OLD.cantitate * :OLD.pretunit * (1 +
                     (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)),
                TVAfact = TVAfact – :OLD.cantitate * :OLD.pretunit *
                    (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
            WHERE nrfact=:OLD.nrfact ;

            -- restul face declansatorul UPD AFTER STATEMENT (trg_lf_upd2)

            --       apoi, se atribuie numarul liniei facturii in care "a ajuns"
            --           si se actualizeaza valoarea si TVA-ul
            :NEW.Linie := pac_vanzari.f_nrlinii (:NEW.NrFact) + 1 ;
            UPDATE fact SET
                NrLinii = NrLinii + 1,
                valtotala = valtotala + :NEW.cantitate *:NEW.pretunit * (1 +
                  (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
                TVAfact = TVAfact + :NEW.cantitate * :NEW.pretunit *
                (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)
            WHERE nrfact=:NEW.nrfact ;

        ELSE
            -- nu s-a modificat nici NrFact, nici Linie
            pac_vanzari.v_nrfact := NULL ;
            pac_vanzari.v_liniestearsa := NULL ;
            UPDATE fact SET
                valtotala = valtotala – :OLD.cantitate *:OLD.pretunit *
                    (1 + (SELECT procTVA FROM produse
                        WHERE codpr=:OLD.codpr)) +
                      :NEW.cantitate *:NEW.pretunit * (1 +
                    (SELECT procTVA FROM produse
                        WHERE codpr=:NEW.codpr)),
                TVAfact = TVAfact – :OLD.cantitate * :OLD.pretunit *
                    (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
            WHERE nrfact=:NEW.nrfact ;
        END IF ;
        pac_vanzari.v_trg_linilfact := FALSE ;
    END IF ;

END ;
/

-------------------------------------------------------------
-- declansatorul de actualizare in LINII_FACT de tip AFTER STATEMENT
```

```
CREATE OR REPLACE TRIGGER trg_lf_upd2 AFTER UPDATE ON linii_fact
BEGIN
    IF pac_vanzari.v_trg_linilfact_del = FALSE THEN
        IF pac_vanzari.v_nrfact IS NULL OR pac_vanzari.v_liniestearsa IS NULL THEN
            -- nu s-a facut trasferul liniei in alta factura, deci stam linistiti
            NULL ;
        ELSE
            pac_vanzari.v_trg_linilfact := TRUE ;
            IF pac_vanzari.f_nrlinii(pac_vanzari.v_nrfact) + 1 =
                    pac_vanzari.v_liniestearsa THEN
                -- linia transferata din factura este ultima
                NULL ;
            ELSE
                -- se scade valoarea atributului LINIE pentru liniile ulterioare celei
                transferare
                pac_vanzari.v_trg_liniifact_del := TRUE ;
                UPDATE linii_fact
                SET linie =  linie – 1
                WHERE NrFact = pac_vanzari.v_nrfact AND linie >=
                pac_vanzari.v_liniestearsa ;
                pac_vanzari.v_trg_liniifact_del := FALSE ;
            END IF ;
            pac_vanzari.v_nrfact := NULL ;
            pac_vanzari.v_liniestearsa := NULL ;
            pac_vanzari.v_trg_liniifact := FALSE ;
        END IF ;
    END IF ;
END;
/
```

Derularea comenzilor şi rezultatele obţinute, aşa după cum sunt prezentate în figura 9.3, ne îndreptăţesc să credem că mecanismul funcţionează. În figură, mai întâi se afişează conţinutul tabelelor FACT şi LINII_FACT pentru facturile cu numerele 3 şi 4. Iniţial, factura 3 conţine trei linii, valoarea sa totală este de 4215500, iar TVA-ul este de 565500, în timp ce factura cu numărul 4 are doar o linie, valoarea sa este de 1190000, iar TVA-ul, 190000. Comanda `UPDATE` pune la încercare declanşatoarele într-una dintre cele mai dificile speţe: linia 1 din factura 3 trece în factura 4. Astfel, pe de o parte, această linie ar trebui să devină linia a doua din factura 4, iar în factura 3, linia a doua ar trebui să devină prima, şi a treia să devină a doua.

Cele două `SELECT`-uri succesoare comenzii `UPDATE` ne arată că, după actualizare, ambele facturi au două linii, valoarea facturii 3 a scăzut de la 4215500 la 3215900, în timp ce valoarea facturii 4 a crescut cu aceeaşi sumă. În plus, ordinea liniilor din tabela LINII_FACT este cea corectă. Ca şi în cazul ştergerii de tupluri din relaţia LINII_FACT, funcţionalitatea declanşatoarelor din *listing*-ul 9.6 este asigurată numai când printr-un `UPDATE` este afectată o singură linie.

```
SQL> SELECT * FROM Fact WHERE NrFact IN (3,4) ;

    NRFACT DATAFACT       CODCL OBS    VALTOTALA    TVAFACT VALINCASATA    NRLINII
---------- --------- ---------- ----- ---------- ---------- ----------- ----------
         4 28-JAN-05       1003          1190000     190000           0          1
         3 28-JAN-05       1002          4215500     565500      800000          3

SQL> SELECT * FROM linii_Fact WHERE NrFact IN (3,4) ;

    NRFACT      LINIE      CODPR  CANTITATE   PRETUNIT
---------- ---------- ---------- ---------- ----------
         4          1          1        800       1250
         3          1          1        700       1200
         3          2          2        800       1600
         3          3          3        900       1700

SQL> update linii_Fact set nrfact=4 where nrfact=3 and linie=1;

1 row updated.

SQL> SELECT * FROM Fact WHERE NrFact IN (3,4) ;

    NRFACT DATAFACT       CODCL OBS    VALTOTALA    TVAFACT VALINCASATA    NRLINII
---------- --------- ---------- ----- ---------- ---------- ----------- ----------
         4 28-JAN-05       1003          2189600     349600           0          2
         3 28-JAN-05       1002          3215900     405900      800000          2

SQL>  SELECT * FROM linii_Fact WHERE NrFact IN (3,4) ;

    NRFACT      LINIE      CODPR  CANTITATE   PRETUNIT
---------- ---------- ---------- ---------- ----------
         4          1          1        800       1250
         4          2          1        700       1200
         3          1          2        800       1600
         3          2          3        900       1700
```

Figura 9.3. Ilustrarea modului de funcţionare a declanşatoarelor

9.5. Relaţii redundante/sintetice

O situaţie ceva mai rar întâlnită este cea în care, din raţiuni de minimizare a timpului de acces, se introduc nu numai atribute suplimentare, redundante, ci chiar relaţii întregi! Chiar dacă economia de joncţiuni şi timpul de răspuns scurtat reprezintă deziderate generale ale tuturor relaţiilor redundante/sintetice, se poate încerca o tipologizare, după cum urmează:

- relaţii necesare implementării unor restricţii mai complexe, greu implementabile (numai) prin clauze CHECK şi declanşatoare;
- relaţii necesare obţinerii mai rapide a unor informaţii punctuale foarte frecvent solicitate;
- relaţii ce vizează accelerarea unor calcule pentru operaţiuni frecvente;
- relaţii-semnal, pentru consemnarea sau semnalarea unor situaţii limită sau a depăşirii unor plafoane etc.;
- relaţii necesare la raportare, uneori chiar relaţii-raport;
- relaţii necesare regrupărilor de date, pentru analize multidimensionale, OLAP etc.;
- relaţii destinate corespondenţelor dintre datele actuale şi cele mai vechi.

Firește, diferenţele dintre aceste categorii nu sunt întotdeauna evidente, destule cazuri pliindu-se pe două sau mai multe situaţii dintre cele enumerate. Prima

categorie a fost ilustrată în paragraful 7.2.3. Pentru a doua categorie, primul exemplu pe care îl avem la îndemână priveşte tot baza de date EXAMENE. Toate notele obţinute de un student sunt contabilizate meticulos în relaţia NOTE (Matricol, CodDisc, DataEx, Nota). Structura este vitală pentru cunoaşterea studenţilor care trebuie să plătească a treia prezentare la un examen, să refacă activitatea din cursul anului (obligatorie la a patra prezentare) etc. Dezavantajul major al acestei scheme este că, în zeci sau chiar sute de cazuri în care interesează media studentului sau numărul său de credite, trebuie luată în calcul doar *ultima notă obţinută la fiecare obiect*!

Lucrul acesta reclamă destul de mult timp, deoarece presupune execuţia unei interogări de genul:

```
SELECT matricol, coddisc, nota
FROM note
WHERE (matricol, coddisc, dataex) IN
   (SELECT matricol, coddisc, MAX(dataex)
    FROM note
    GROUP BY matricol, coddisc)
```

care, vă asigur, este costisitoare când NOTE are sute de mii sau milioane de înregistrări.

Deşi pare greu acceptabilă, putem propune soluţia folosirii unei relaţii speciale, NOTE_FINALE (Matricol, CodDisc, NotaFinală), în care fiecare student are, pentru fiecare disciplină la care a susţinut măcar un examen, numai ultima notă (cea finală), obţinută la prima, a doua... prezentare (vezi *listing*-ul 9.7 de pe Web). La inserarea unei linii în tabela NOTE, inserare care respectă ordinea cronologică a examinărilor, trebuie verificat dacă aceasta ar fi prima examinare de la disciplina respectivă, caz în care trebuie făcută o inserare în NOTE_FINALE, iar în caz contrar trebuie făcută modificarea valorii atributului NOTE_FINALE.notafinală – vezi *listing*-ul 9.8.

Listing 9.8. Declanşatorul de inserare în NOTE

```
CREATE OR REPLACE TRIGGER trg_note_ins
   AFTER INSERT ON note FOR EACH ROW
BEGIN
   UPDATE note_finale SET notafinala = :NEW.nota
   WHERE matricol=:NEW.matricol AND coddisc=:NEW.coddisc ;

   IF SQL%ROWCOUNT = 0 THEN -- nr. liniilor modificate este 0
         INSERT INTO note_finale VALUES (:NEW.matricol, :NEW.coddisc, :NEW.nota) ;
   END IF;
END ;
```

Marele pericol al soluţiei decurge din supoziţia că operarea notelor respectă ordinea cronologică a susţinerii examenelor. Dacă, din neatenţie, se preiau notele din restanţe înaintea celor din sesiune, corectitudinea datelor este compromisă. Aşa

încât merită să introducem în NOTE_FINALE şi atributul `DataUltimuluiExamen`, care să ne scutească de orice emoţie: NOTE_FINALE (Matricol, CodDisc, NotaFinală, DataUltimuluiExamen):

```
ALTER TABLE note_finale ADD (DataUltimuluiExamen DATE)
```

Iată cum ar trebui să arate noul declanşator de inserare (*listing*-ul 9.9):

Listing 9.9. Noul declanşator de inserare în NOTE

```
CREATE OR REPLACE TRIGGER trg_note_ins
   AFTER INSERT ON note FOR EACH ROW
BEGIN
   UPDATE note_finale
   SET notafinala = :NEW.nota, dataultimululexamen = :NEW.dataex
   WHERE matricol=:NEW.matricol AND coddisc=:NEW.coddisc
      AND dataultimululexamen < :NEW.dataex ;

   IF SQL%ROWCOUNT = 0 THEN
      -- inserarea se face numai daca nu e nici o inregistrare pentru
      --  examinarea (matricol, coddisc) curenta în NOTE_FINALE
      INSERT INTO note_finale
         SELECT matricol, :NEW.coddisc,  :NEW.nota, :NEW.dataex
         FROM studenti WHERE matricol=:NEW.matricol AND
            NOT EXISTS
             (SELECT 1 FROM note_finale WHERE matricol=:NEW.matricol AND
                coddisc=:NEW.coddisc) ;
   END IF;
END ;
```

Dacă exemplul vi s-a părut cu totul exagerat, încercăm un altul: ce ziceţi de o bază de date pentru gestionarea informaţiilor referitoare la sezonul curent din Formula 1? Sezonul este alcătuit dintr-o serie de curse desfăşurate pe diferite circuite (Estoril, Monza, Silverstone etc.). O poziţie în primele 8 locuri oferă un anumit punctaj: poziţia 1 conferă 10 puncte, poziţia 2 – 8 puncte, 3 – 6 puncte, 4 – 5 puncte..., 8 – 1 punct. Graful DF este cel din figura 9.4.

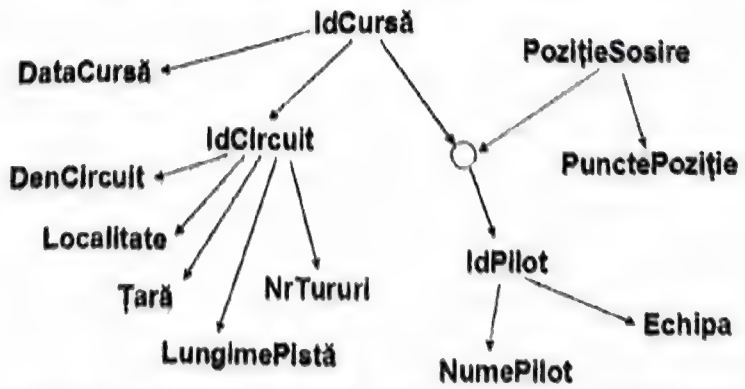


Figura 9.4. Graful DF pentru baza de date FORMULA1

Decupăm relaţiile din graf (scriptul de creare este disponibil pe Web sub forma *listing*-ului 9.10, iar cel de populare este conţinut în *listing*-ul 9.11):

CIRCUITE (IdCircuit, DenCircuit, Localitate, Ţară, LungimePistă, NrTururi)
PILOŢI (IdPilot, NumePilot, Echipa)
LOCURI_PUNCTE (PoziţieSosire, PunctePoziţie)
CURSE (IdCursă, DataCursă)
REZULTATE_CURSE (IdCursă, PoziţieSosire, IdPilot)

Dintre informaţiile cel mai frecvent solicitate, am putea lua în discuţie cel puţin două. Prima este: *ce număr de puncte are pilotul P1* sau *echipa E1*? Folosind schema de mai sus, normalizată cum e ea, trebuie operate, de fiecare dată, joncţiunile:

```
SELECT SUM(PunctePoziţie) AS NrPuncte
FROM locuri_puncte lp
INNER JOIN rezultate_curse rc
ON lp.PoziţieSosire=rc.PoziţieSosire
   INNER JOIN piloţi p ON rc.IdPilot=p.IdPilot
WHERE NumePilot = 'P1'
```

Nu este, astfel, prea îndrăzneaţă ideea de a introduce în relaţia REZULTATE_CURSE atributul NrPuncteCursă:

```
ALTER TABLE rezultate_curse ADD (NrPuncteCursa NUMBER(3)) ;
```

Acest nou atribut va fi actualizabil prin *trigger*-ele de inserare şi modificare ale tabelei REZULTATE_CURSE, după modelul din *listing*-ul 9.12:

Listing 9.12. Calculul automat al numărului de puncte obţinut pentru fiecare clasare

```
CREATE OR REPLACE TRIGGER trg_rc_ins
   BEFORE INSERT ON rezultate_curse FOR EACH ROW
DECLARE
   v_puncte NUMBER(2) := 0 ;
BEGIN
   SELECT PunctePozitie INTO v_puncte FROM locuri_puncte
   WHERE PozitieSosire = :NEW.PozitieSosire ;

   :NEW.NrPuncteCursa := v_puncte ;
END ;
```

Mergând mai departe, n-ar fi o idee tocmai rea ca în tabela PILOŢI să centralizăm punctele obţinute pe întreg sezonul printr-un atribut special dedicat – PuncteSezon. Gestiunea sa este oarecum banală, realizându-se prin intermediul declanşatoarelor tabelei REZULTATE_CURSE, mai precis, a atributului NrPuncteCursă. În acest

fel am fi în măsură să obţinem în orice moment clasamentul piloţilor. Pentru a avea un clasament *online* pe echipe, soluţia cea mai la îndemână ar fi o tabelă virtuală:

```
CREATE VIEW puncte_echipe
   SELECT echipa, SUM(PuncteSezon) AS PuncteEchipă
   FROM piloţi
   GROUP BY echipe
```

Amatorii de senzaţii tari pot folosi însă şi o relaţie specială:

PUNCTE_ECHIPE {Echipa, Puncte}

ale cărei înregistrări sunt gestionate exclusiv prin intermediul declanşatoarelor relaţiei PILOŢI. Această relaţie acoperă, într-o măsură mai mare sau mai mică, atât categoria celor dedicate obţinerii rapide a unor informaţii frecvent solicitate, cât şi categoria relaţiilor-raport. Rămâne să vă încordaţi forţele pe cont propriu, deoarece noi sărim cu discuţia la alte cazuri.

Pentru exemplificarea tabelelor redundante/sintetice destinate accelerării calculelor în operaţiuni complexe, revenim la paragraful 8.3, care s-ar fi vrut dedicat grafurilor, dar a sfârşit doar prin a contura o schemă de baze de date utilă unei firme de transport intern de călători dotată cu celebrele microbuze cu minunaţii lor şoferi. Structura tabelelor este ilustrată în figura 8.23, iar *listing*-ul 8.14, vecin figurii, propunea o funcţie (f_cost_bilet) ce primeşte trei parametri, localitatea de urcare a călătorului, cea de coborâre şi identificatorul rutei urmate de mijlocul de transport, şi furniza, după cum îi spune şi numele, contravaloarea biletului (presupunând că şoferul n-ar face un mic gheşeft, în dulcele spirit balcanic).

Deşi poate interesantă, funcţia mănâncă ceva timp şi resurse, deoarece la fiecare bilet eliberat se parcurge cursorul în care se află încărcate toate localităţile de pe ruta specificată aflate între localităţile de urcare şi de coborâre, adunându-se numărul de kilometri şi, la final, determinându-se costul călătoriei. Având în vedere că recalcularea (scumpirea, de fapt) biletelor este din ce în ce mai rară, o idee nu tocmai rea ar fi să se recurgă la o relaţie suplimentară, altminteri destul de banală:

TARIFE {IdRută, Loc1, Loc2, Preţ}

Înregistrările acestei tabele pot fi generate printr-o procedură care va apela şi funcţia din *listing*-ul 8.14, iar după generare, întregul calcul va fi înlocuit cu accesul, incomparabil mai rapid, la una dintre înregistrările acestei noi tabele. Unde mai punem că această nouă tabelă nici nu (prea) trebuie modificată până la următoarea scumpire a biletelor sau adăugarea de noi rute.

Cei care aţi avut răbdare să citiţi până spre capăt capitolul 8 aţi sesizat că nici în privinţa rezervărilor de locuri la cursele firmei nu ne simţeam prea bine. Declanşatorul relaţiei REZERVĂRI făcea un adevărat tur de forţă încercând să determine,

pentru cursa respectivă, vreun segment de rută pentru care numărul biletelor rezervate depăşeşte numărul locurilor microbuzului. Şi în această situaţie putem îndulci rezolvarea creând o relaţie suplimentară/redundantă/sintetică ş.a.m.d:

CURSE_REZERVĂRI {IdCursă, Loc1, Loc2, NrLocuriRez}

Actualizată prin declanşatoarele tabelei REZERVĂRI, această nouă relaţie poate simplifica şi accelera semnificativ urmărirea locurilor disponibile pentru fiecare cursă.

O categorie mai exotică de relaţii sintetice, negăsită (de mine) în materialele dedicate denormalizării, este cea a tabelelor-semnal, apropiată, ca finalitate, de tabelele jurnal pe care le vom prezenta pe scurt în ultimul capitol. Dacă tot n-am apucat să ne despărţim de aplicaţia pentru firma de transport călători, am putea imagina o tabelă în care să consemnăm momentul în care, pentru o cursă, numărul de locuri rezervate depăşeşte capacitatea microbuzului, astfel încât, fie se sistează rezervările (cazul descris în declanşatorul din *listing*-ul 8.15), fie se apelează la un alt microbuz, mai mare (adică *macrobuz*), fie se primesc oricâte rezervări şi, în funcţie de acestea, se suplimentează cursa cu încă unul sau mai multe mijloace de transport:

DEPĂŞIRI_LOCURI {IdDepăşire, IdCursa, NrLocuriDepăşire}

Exemplul nu este chiar mişcător, aşa că am putea încerca un altul. În privinţa unei aplicaţii pentru gestiunea stocurilor, există firme, nu prea numeroase, care operează după următorul scenariu: unele materii prime/materiale/mărfuri sunt achiziţionate de la furnizori plasaţi la oarecare distanţă. Deoarece durata procesului de aprovizionare poate pune firma în situaţii de blocaje, se ia decizia ca fiecărui sortiment să-i fie stabilit un stoc de siguranţă, iar aplicaţia dedicată stocurilor să semnaleze momentul în care stocul curent al materialului respectiv scade sub nivelul celui de siguranţă.

MATERIALE {IdSortiment, DenSortiment, UM, ContClasa3, Grupa, StocSiguranţă}
...
INTRĂRI_MATERIALE {IdIntrare, IdSortiment, CantIntrată, PreţIntrare}
...
IEŞIRI_MATERIALE {IdIeşire, IdSortiment, CantIeşită, CostDescărcare}
SEMNALE {IdSemnal, DataOraSemnal, IdSortiment, CantitateSubStocSiguranţă}

Cei care se ocupă cu aprovizionarea nu ar avea decât să interogheze periodic această tabelă sau să fie atenţionaţi într-un fel sau altul la fiecare inserare de înregistrări în tabela SEMNALE.

O altă categorie enumerată la începutul acestui paragraf făcea referinţă la relaţiile sintetice de tip raport. O asemenea tabelă poate fi considerată PUNCTE_ECHIPE {Echipa, Puncte} din exemplul curselor de Formula 1, deşi prezentarea sa a fost făcută la categoria relaţiilor redundante/sintetice pentru mărirea vitezei de furnizare a informaţiilor frecvent solicitate.

Şi aici putem vorbi de două situaţii, una în care rapoartele pot fi obţinute destul de uşor, creând o tabelă virtuală (*view*) pe scheletul machetei listei, în timp ce a doua chiar necesită tabele în toată regula, cu tot cu redundanţa lor. După exemplul tabelei virtuale PUNCTE_ECHIPE de aproximativ două pagini mai sus, ne reîntoarcem la figura 9.1 şi la discuţiile din preajma ei. Ţinând seama de frecvenţa accesării soldurilor şi rulajelor, am apelat, pentru tabela CONTURI_ELEMENTARE, la două atribute noi, RulajDB şi RulajCR, actualizabile exclusiv prin declanşatoarele tabelei DETALII_ OPERAŢIUNI (vezi şi paragraful 7.3). Un raport solicitat frecvent este balanţa de verificare, din care o porţiune este prezentată în figura 9.5[26].

CONT	DENUMIRECONT	T	SOLDINITDB	SOLDINITCR	RULAJDB	RULAJCR	TOTALSUMEDB	TOTALSUMECR
101	Capital social	P	0	0	0	0	0	0
1011	Capital social subscris ne-varsat	P	0	0	0	0	0	0
1012	Capital social subscris varsat	P	0	0	0	0	0	0
121	Profit si pierdere	B	0	0	375000	0	375000	0
211	Terenuri si amenajari	A	0	0	11900000	1900000	11900000	1900000
2111	Terenuri	A	0	0	11900000	0	11900000	0
2112	Amenajari	A	0	0	0	1900000	0	1900000
301	Materii prime	A	0	0	4000000	375000	4000000	375000
308	Diferente de pret la materii prime	B	0	0	0	0	0	0
401	Furnizori	P	0	0	0	16660000	0	16660000
411	Clienti	A	0	0	0	0	0	0
428	Alte creante si datorii fata de salariati	B	0	0	0	0	0	0
4281	Alte creante fata de salariati	A	0	0	0	0	0	0
4282	Alte datorii fata de salariati	P	0	0	0	0	0	0
4426	TVA deductibila	A	0	0	2660000	0	2660000	0
4427	TVA colectata	P	0	0	0	0	0	0
601	Cheltuieli cu materii prime	A	0	0	375000	375000	375000	375000
------	--- T O T A L	-	0	0	19310000	19310000	19310000	19310000

Figura 9.5. O porţiune dintr-o balanţă de verificare

Dificultatea ţine de faptul că soldurile iniţiale şi operaţiunile contabile sunt introduse la nivel de conturi elementare, în timp ce balanţa se obţine doar la nivelul conturilor sintetice de gradurile 1 şi 2. Se poate concepe o tabelă specială în acest scop – BALANŢĂ {SimbolCont, DenumireCont, TipCont, SoldInitDB, SoldInitCR, RulajDB, RulajCR, TotalSumeDB, TotalSumeCR, SoldFinalDB, SoldFinalCR} –, în care actualizările să se facă automat, din declanşatoarele tabelei CONTURI_ELEMENTARE. Efortul de scriere a pachetelor şi declanşatoarelor n-ar fi prea mare, însă soluţia este cam penibilă, întrucât, ori de câte ori s-ar dori listat conţinutul balanţei curente, ar fi suficientă o comandă SELECT de genul:

```
SELECT SimbolCont AS Cont, DenumireCont, TipCont,
   (SELECT NVL(SUM(SoldInitDB),0) FROM conturi_elementare
    WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
      = pc.SimbolCont) SoldInitDB,
```

26. Din raţiuni de lizibilitate, în figură nu au fost incluse ultimele două coloane ale balanţei, care conţin soldurile finale debitoare şi creditoare.

```
(SELECT NVL(SUM(SoldInitCR),0) FROM conturi_elementare
 WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
  = pc.SimbolCont) SoldInitCR,
(SELECT NVL(SUM(RulajDB),0) FROM conturi_elementare
 WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
  = pc.SimbolCont) RulajDB,
(SELECT NVL(SUM(RulajCR),0) FROM conturi_elementare
 WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
  = pc.SimbolCont) RulajCR,
(SELECT NVL(SUM(SoldInitDB),0)+ NVL(SUM(RulajDB),0)
 FROM conturi_elementare
 WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
  = pc.SimbolCont) TotalSumeDB,
(SELECT NVL(SUM(SoldInitCR),0)+ NVL(SUM(RulajCR),0)
 FROM conturi_elementare
 WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
  = pc.SimbolCont) TotalSumeCR,
(SELECT NVL(SUM(
  CASE TipCont
  WHEN 'A' THEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
    - NVL(RulajCR ,0)
  WHEN 'P' THEN 0
  ELSE
    CASE WHEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
      - NVL(RulajCR ,0) < 0
    THEN 0 ELSE NVL(SoldInitDB,0) + NVL(RulajDB,0)
       - NVL(RulajCR,0)
    END
  END), 0)
FROM conturi_elementare
WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
  = pc.SimbolCont) SoldFinalDB,
(SELECT NVL(SUM(
  CASE TipCont
  WHEN 'P' THEN NVL(SoldInitCR,0) + NVL(RulajCR,0)
    - NVL(RulajDB ,0)
  WHEN 'A' THEN 0
  ELSE
    CASE WHEN NVL(SoldInitCR,0) + NVL(RulajCR,0)
      - NVL(RulajDB ,0) < 0
    THEN 0 ELSE NVL(SoldInitCR,0) + NVL(RulajCR,0)
       - NVL(RulajDB,0)
    END
  END), 0)
FROM conturi_elementare
```

```
   WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
     = pc.SimbolCont) SoldFinalCR
FROM plan_conturi pc
WHERE LENGTH(SimbolCont)=4 OR LENGTH(SimbolCont)=3
UNION
SELECT '=====', '=== T O T A L ', '=',
   (SELECT NVL(SUM(SoldInitDB),0) FROM conturi_elementare),
   (SELECT NVL(SUM(SoldInitCR),0) FROM conturi_elementare),
   (SELECT SUM(RulajDB) FROM conturi_elementare),
   (SELECT SUM(RulajCR) FROM conturi_elementare),
   (SELECT NVL(SUM(SoldInitDB),0)+ NVL(SUM(RulajDB),0)
      FROM conturi_elementare),
   (SELECT NVL(SUM(SoldInitCR),0)+ NVL(SUM(RulajCR),0)
    FROM conturi_elementare),
   (SELECT NVL(SUM(
     CASE TipCont
     WHEN 'A' THEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
       - NVL(RulajCR ,0)
     WHEN 'P' THEN 0
     ELSE
       CASE WHEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
         - NVL(RulajCR ,0) < 0
       THEN 0 ELSE NVL(SoldInitDB,0) + NVL(RulajDB,0)
           - NVL(RulajCR,0)
       END
     END), 0)
     FROM conturi_elementare ce INNER JOIN plan_conturi pc2
       ON contelementar=pc2.simbolcont),
     (SELECT NVL(SUM(
       CASE TipCont
       WHEN 'P' THEN NVL(SoldInitCR,0) + NVL(RulajCR,0)
         - NVL(RulajDB ,0)
       WHEN 'A' THEN 0
       ELSE
         CASE WHEN NVL(SoldInitCR,0) +
          NVL(RulajCR,0)- NVL(RulajDB ,0) < 0
         THEN 0
         ELSE NVL(SoldInitCR,0) + NVL(RulajCR,0)
          - NVL(RulajDB,0)
         END
       END), 0)
     FROM conturi_elementare ce INNER JOIN plan_conturi pc2
       ON contelementar=pc2.simbolcont)
FROM dual
```

Pentru cei mai puţin răbdători, fraza supraponderală a fost salvată în *listing*-ul 9.13 (pe Web), iar eliminarea neplăcerii de a-i admira „rotunjimile" la lansare se poate rezolva prin crearea unei tabele virtuale care va fi consultată ulterior cu un `SELECT` obişnuit:

```
CREATE VIEW view_balanta AS
SELECT.....
```

Pentru unele rapoarte mai sofisticate, cum sunt fişele de cont, fişele-şah (tot pentru conturi) etc., argumentele creării unor tabele speciale sintetice sunt ceva mai uşor de luat în seamă. De exemplu, pentru obţinerea fişelor de cont se poate apela la o tabelă redundantă de genul:

FIŞE_CONT {`SimbolCont`, `DenumireCont`, `Debit_Credit`, `SoldIniţial`, `Cont_Corespondent`, `Rulaj`}

Actualizarea unei asemenea tabele nu ar costa prea mult, iar avantajele ar fi ceva mai vizibile decât în cazul balanţei de verificare.

Mergând către finalul (semi)clasificării noastre din acest paragraf, vom zăbovi preţ de câteva fraze la relaţiile suplimentare necesare regrupărilor de date, analizelor multidimensionale etc. Un prim exemplu ar viza analiza vânzărilor. În relaţia PRODUSE {`CodPr`, `DenPr`, `UM`, `ProcTVA`, `Grupa`} există un atribut – `Grupa` – prin care specificăm dacă produsul face parte din categoria mărfurilor *alimentare* sau *tutun*, *dulciuri* etc. La un moment dat, ne-ar putea interesa ponderea vânzărilor de mărfuri alimentare din carne de porc faţă de cele de vită sau de pui. Sau care ar fi tipul de cafea decofeinizată cel mai bine vândut. Pentru a răspunde la acest gen de probleme putem defini un soi de ierarhie a mărfurilor, după ideea aplicată competenţelor necesare posturilor dintr-o organizaţie (vezi capitolul 8).

Astfel, în categoria *băuturi* am include *răcoritoarele*, *berea*, *vinul*, *apa minerală* şi *băuturile spirtoase*. *Berea*, ca să luăm una dintre subcategorii, am împărţi-o în *neagră/brună* şi *blondă*, iar tipurile de *bere blondă* le-am descompune pe *mărci* (Carlsberg, Beck's, Timişoreana – asta-i reclamă nemascată, nu-i aşa?), apoi pe *forme de prezentare* (sticle, cutii, PET-uri), apoi pe volumul (număr de centilitri) de bere conţinut. Am putea avea o relaţie:

CATEGORII {`IdCategorie`, `Dencategorie`, `Caracteristică`, `MărimeaCaracteristicii`, `IdCategorieSuperioară`}

iar în PRODUSE, în loc de `Grupa`, am folosi atributul-cheie străină `IdCategorie`. Ei bine, ce facem dacă vrem să comparăm vânzările de bere la *cutie*, faţă de cea la *sticlă* şi cea la *PET*, indiferent de producător? Mai general spus, ne interesează categorii de sortimente care sunt plasate pe mai multe ramuri ale clasificaţiei arborescente. Soluţia care ne poate rezolva problema este de a defini o tabelă de criterii şi alta care să conţină toate categoriile incluse în criteriul respectiv:

CRITERII {IdCriteriu, DenCriteriu, Observaţii}

şi

CRITERII_CATEGORII {IdCriteriu, IdCategorie}.

Astfel, în CRITERII vom insera o linie pentru criteriul *Bere la cutie*, pentru care în CRITERII_CATEGORII va exista câte o linie pentru fiecare sortiment de bere la cutie existent: *bere Timişoareana Gold - cutie 500 ml*; *bere Timişoareana Pils - cutie 500 ml*; *bere Guiness - cutie 500 ml* etc.; apoi, criteriul *Bere la sticlă* ş.a.m.d.

Un al doilea exemplu vine din aplicaţia de CONTABILITATE. În PLAN_CONTURI, nomenclatorul general al conturilor a fost completat cu analitice definite de organizaţie. Astfel, putem defini în clasa 3 analitice pentru fiecare tip de materie primă/material: 301.001 pentru nisip, 301.002 pentru var, 301.003 pentru ciment etc. Conturile de cheltuieli din clasa 6 le-am putea grupa după obiectivele în lucru şi natura operaţiilor derulate, o serie de operaţii fiind comune mai multor obiective:

- 601.001.001 - cheltuieli la obiectivul nr. 1, operaţiunea 1;
- 601.001.003 - cheltuieli la obiectivul nr. 1, operaţiunea 3;
- 601.003.001 - cheltuieli la obiectivul nr. 3, operaţiunea 1.

La orice consum de materie primă/material se debitează contul din clasa 6 ce indică obiectivul şi operaţia destinaţie şi se creditează analiticul ce indică natura materialui consumat. Putem, astfel, afla toate cheltuielile cu materiale pentru fiecare obiectiv în parte, ca şi modul în care au fost consumate nisipul, varul, cimentul pe obiective. Însă consumul de materiale pe operaţiuni (pentru toate obiectivele) este ceva mai este greu de aflat, iar dacă aceeaşi operaţiune are analitic diferit de la obiectiv la obiectiv, greul devine imposibil. Soluţia ar fi crearea unei relaţii noi:

CHELTUIELI_OBIECTIVE_OPERAŢIUNI {ContAnalitic, Obiectiv, Operaţiune}

Atenţie, însă! Nu cred că ar fi nimerit să asimilăm aceste artificii denormalizării propriu-zise, întrucât relaţiile introduse aduc informaţii noi, greu, dacă nu chiar imposibil de obţinut printr-un alt artificiu. Aşa că, în loc de denormalizare, mai bine vorbim de postnormalizare, că sună mai bine!

Ultima categorie de relaţii redundant-sintetice dintre cele enumerate o rezervăm pentru ultimul subcapitol.

9.6. Atribute introduse pe baza grupurilor repetitive

În capitolul 1 am discutat diverse soluţii de eliminare a grupurilor repetitive şi atomizare a valorilor tuturor atributelor din bază, chiar dacă teoreticienii modelului relaţional au eliminat dintre poruncile relaţionalului atomicitatea atributelor, aşa cum a fost înţeleasă de majoritatea autorilor/practicienilor, drept „scalaritatea"

valorilor dintr-un domeniu. Ca idee generală, recomandam ca eliminarea grupurilor repetitive să fie făcută în funcţie de specificul aplicaţiei sau chiar să nu fie făcută deloc.

Exemplu

În urma normalizării, ţinând seama de faptul că, pe de o parte, o firmă client poate avea mai multe telefoane de contact sau adrese e-mail, iar, pe de altă parte, nu există două numere de telefon sau adrese de e-mail absolut identice, s-ar putea ajunge la o structură de genul:

CLIENŢI {CodCl, DenCl, CodFiscal, StradaCl, NrStadaCl, BlocScApCl, CodPost}
TELEFOANE_CLIENŢI {Telefon, CodCl}
E_MAIL_CLIENŢI {eMail, CodCl}

Fără nici un dubiu, schema este scrupulos normalizată, neexistând atribute compuse. Cu toate acestea, nici `Telefon`, nici `eMail` nu sunt atribute care să fie supuse operaţiunilor de agregare (numărare, însumare) şi, mai important, nu participă nici în joncţiuni cu alte tabele ale bazei de date. Este motivul pentru care, în acest gen de situaţii, se poate lua fără prea multe remuşcări decizia acceptării valorilor compuse pentru atributele `Telefon` şi `eMail`:

CLIENŢI3 {CodCl, DenCl, CodFiscal, StradaCl, NrStadaCl, BlocScApCl, CodPost, Telefoane, eMailuri}.

Putem astfel vorbi de un soi de denormalizare, de revenire la o formă pre-1FN. Vestea bună e că tipurile de date compozite pot fi gestionate în multe SGBD-uri fără eforturi deosebite. Spre exemplu, Oracle prezintă două tipuri de colecţii ce pot fi stocate într-o bază de date, *tabele incluse* (imbricate), în original *nested tables*, şi *vectori de mărime variabilă – VARRAYS*. Tocmai pentru ca exemplul de mai sus să prindă viaţă, în *listing*-ul 9.14 se creează două tipuri de tabele incluse, unul pentru numere de telefon şi altul pentru adresele de e-mail. Aceste tipuri sunt folosite în tabela CLIENŢI3.

Listing 9.14. Valori neatomice folosind tabele incluse (*nested tables*) în Oracle

```
DROP TABLE clienti3;
DROP TYPE nt_adrese_email ;
DROP TYPE nt_telefoane ;

CREATE TYPE nt_telefoane AS TABLE OF CHAR(11)
/

CREATE TYPE nt_adrese_email AS TABLE OF VARCHAR2(30)
/

CREATE TABLE CLIENTI3 (
    CodCl NUMBER(6) PRIMARY KEY,
```

```
    DenCl VARCHAR2(40),
    CodFiscal CHAR(8),
    StradaCl VARCHAR2(30),
    NrStadaCl VARCHAR2(7),
    BlocScApCl VARCHAR2(25),
    CodPost NUMBER(6),
    Telefoane nt_telefoane,
    eMailuri nt_adrese_email
    )
NESTED TABLE telefoane STORE AS telefoane_tab,
NESTED TABLE eMailuri STORE AS eMailuri_tab
/

INSERT INTO clienti3 VALUES (1001, 'Client 1', 'R133303', 'Sapientei',
    '22bis', NULL, 710710,
    nt_telefoane('0232344444', '0788881919', '0744444444'),
    nt_adrese_email('client1@k.ro', 'client.1@yahoo.com')
    );
INSERT INTO clienti3 VALUES (1002, 'Client 2', 'R1344533', 'Pacientei',
    '13bis', 'Bl.H, Sc.B, Ap.5', 710705,
    nt_telefoane('0239344554', '0723881919', '0722444444', '0744456789'),
    nt_adrese_email('client2@hotmail.ro', 'client2@yahoo.com')
    );
```

Tabela CLIENŢI3 este populată cu două înregistrări, primul client având trei numere de telefon şi două adrese e-mail, iar al doilea, patru numere de telefon şi tot două adrese de e-mail. Conţinutul tabelei este afişat după cum arată figura 9.6. „Relaţionalizarea" conţinutului tabelei incluse presupune folosirea expresiei `TABLE` plasată în clauza `FROM` a unei fraze `SELECT`. Astfel,

```
SELECT *
FROM TABLE (SELECT c.telefoane FROM clienti3 c
     WHERE c.codcl=1002) x ;
```

afişează telefoanele celui de-al doilea client (vezi figura 9.6), în timp ce:

```
SELECT c.dencl, t.*
FROM clienti3 c, TABLE(c.telefoane) t WHERE codcl=1002 ;
```

le afişează precedându-le de numele clientului.

Pentru a de-denormaliza, adică a renormaliza telefoanele şi adresele e-mail, soluţia este relativ simplă:

```
SELECT c.codcl, t.*
FROM clienti3 c, TABLE(c.telefoane) t  ;
```

respectiv:

```
SQL> SELECT * FROM clienti3 c ;

     CODCL DENCL           CODFISCA STRADACL       NRSTADA BLOCSCAPCL           CODPOST
---------- --------------- -------- --------------- ------- ------------------ ----------
TELEFOANE
--------------------------------------------------------------------------------
EMAILURI
--------------------------------------------------------------------------------
      1001 Client 1        R133303  Sapientei      22bis                        710710
NT_TELEFOANE('0232844444 ', '0788881919 ', '0744444444 ')
NT_ADRESE_EMAIL('client1@k.ro', 'client.1@yahoo.com')

      1002 Client 2        R1344533 Pacientei      13bis   Bl.H, Sc.B, Ap.5     710705
NT_TELEFOANE('0239844554 ', '0723881919 ', '0722444444 ', '0744456789 ')
NT_ADRESE_EMAIL('client2@hotmail.ro', 'client2@yahoo.com')


SQL> SELECT *
  2  FROM TABLE (SELECT c.telefoane FROM clienti3 c
  3    WHERE c.codcl=1002) x ;

COLUMN_VALU
-----------
0239844554
0723881919
0722444444
0744456789
```

Figura 9.6. Gestionarea tabelelor incluse

```
SELECT c.codcl, t.*
FROM clienti3 c, TABLE(c.emailuri) t  ;
```

Întrucât aflarea numerelor de telefon ale unui client nu ridică probleme, să vedem cum se soluţionează problema: *care dintre clienţi are numărul 0722444444 ?* Dacă ţinem cont că numele automat al atributului obţinut prin aplicarea funcţiei `TABLE` este `column_value`, soluţia este mai mult decât banală:

```
SELECT c.dencl, t.*
FROM clienti3 c, TABLE(c.telefoane) t
WHERE t.column_value='0722444444'
```

În acest mod stocăm atributele multivaloare de o manieră ceva mai „naturală", în timp ce pentru a obţine informaţii continuăm să apelăm la frazele SELECT în maniera obişnuită de interogare în SQL.

Contraexemplu

Pentru baza de date dedicată gestionării cărţilor dintr-o bibliotecă avem şi un contraexemplu de folosire a grupurilor repetitive. Astfel, fiecare carte are unul, doi sau mai mulţi autori, aşa încât ar fi fost tentantă o structură de genul:

CĂRŢI {ISBN, Titlu, Autori, ...}

Spre deosebire de cazul telefoanelor şi adreselor de e-mail, nu este recomandabil ca atributul `Autori` să fie declarat de tip colecţie, şi aceasta din trei motive. Pe de o

parte, căutările după un autor sau doi autori etc. sunt relativ numeroase, iar asemenea gen de căutare folosind opţiuni SQL ar fi îngreunată în condiţiile folosirii colecţiilor. În al doilea rând, este foarte posibil ca o aplicaţie „profesională" să folosească o tabelă-nomenclator AUTORI, astfel încât postura atributului `Autori` ar fi de cheie străină care va apărea, inevitabil, în joncţiuni. De asemenea, un al treilea motiv ţine de faptul că *ordinea* autorilor este foarte importantă; la afişarea datelor despre o carte este neapărat ca ordinea autorilor să fie cea de pe copertă. Aşa încât cea mai indicată variantă foloseşte un atribut special pentru a arăta ordinea fiecărui autor pe coperta unei cărţi.

Iar dacă, pentru un raport, se doreşte afişarea bibliografică „clasică": autori (în ordinea de pe copertă), titlu, editură/publicaţie, anul apariţiei, se poate crea o tabelă virtuală, astfel încât valoarea atributului `Autori` să fie „calculată" cu o funcţie specială care să returneze un şir de caractere de lungime variabilă, în funcţie de câţi autori au participat la scrierea respectivului material.

9.7. Ruperea relaţiilor pentru creşterea vitezei de acces

Una dintre poruncile multor autori importanţi în domeniul proiectării bazelor de date este: *nu rupeţi (descompuneţi) relaţiile deplin normalizate în relaţii mai mici!*[27] Ca buni români, trebuie să privim orice lege cu rezervele de rigoare, căutând porţiunile mai laxe. Revenim la relaţia STUDENŢI. Fără a complica inutil discuţia, în capitolele şi paragrafele precedente am invocat doar câteva informaţii care interesează secretariatul unei facultăţi. Pentru o aplicaţie dedicată gestiunii datelor despre studenţi şi activităţi didactice, discuţia trebuie extinsă serios, deoarece documentele ce trebuie întocmite sunt dintre cele mai diverse:

- pentru cazările în căminele universităţii este necesară o listă a studenţilor pe sexe (băieţi/fete), ani de studii şi specializări, listă în care să nu fie incluşi cei din oraşul Iaşi (e de presupus că aceştia stau cu părinţii);
- centrele militare judeţene solicită în fiecare an o listă cu băieţii care sunt studenţi ai facultăţii (pentru a nu întocmi pentru ei ordine de încorporare);
- dosarul fiecărui student trebuie să cuprindă şi date legate de liceul absolvit, anul absolvirii, media de la Bacalaureat şi media pe cei patru ani de liceu;
- în foile matricole şi alte documente legate de şcolaritate apar: numele dinaintea căsătoriei (adică numele din certificatul de naştere, pentru persoanele care în urma căsătoriei şi-au schimbat numele de familie), numele mamei şi al tatălui, data şi locul naşterii etc.

Chiar dacă ne oprim aici cu discuţia, ar însemna că tabela STUDENŢI ar avea o structură de genul: STUDENŢI (`Matricol`, `NumePren`, `Sex`, `An`, `Modul`, `IdSpec`, `Grupa`, `Adresa_Str`, `Adresa_Nr`, `Adresa_Bloc`, `Adresa_Scară`, `Adresa_Etaj`, `Adresa_Apart`, `Adresa_CodPoştal`, `NumeCertificatNaştere`, `DataNaşterii`,

27. Vezi, spre exemplu, [Fleming & von Halle 89], p. 195.

LoculNaşterii, JudeţulNaşterii, Mama, Tata, Liceu_Absolvit, CodPoştal_Liceu, Medie_Bac, Medie_Ani_Liceu, StagiuMilitar}. Iar dacă ne mai concentram puţin, mai găseam nişte informaţii suplimentare. Toate atributele sunt în depedenţă funcţională faţă de Matricol, aşa că includerea lor în aceeaşi relaţie este întru totul legitimă.

Cu toate acestea, din momentul preluării acestor date, în anul I, până după momentul absovirii (şi, implicit, arhivării datelor), dintre toate aceste atribute doar câteva sunt folosite frecvent, cea mai mare parte fiind utile doar de câteva ori. De aceea, atunci când numărul studenţilor este de ordinul miilor sau zecilor de mii (cazul unei universităţi), numărul disciplinelor poate fi de câteva sute, iar numărul de note preluate se poate ridica la sute de mii, chiar milioane, n-ar fi rău deloc ca relaţia STUDENŢI să se spargă astfel:

STUDENŢI {Matricol, NumePren, An, Modul, IdSpec, Grupa}
STUD_DATEGENERALE {Matricol, Sex, Adresa_Str, Adresa_Nr, Adresa_Bloc, Adresa_Scară, Adresa_Etaj, Adresa_Apart, Adresa_CodPoştal, NumeCertificatNaştere, DataNaşterii, LoculNaşterii, JudeţulNaşterii, Mama, Tata, StagiuMilitar}
STUD_LICEU {Matricol, Liceu_Absolvit, CodPoştal_Liceu, Medie_Bac, Medie_Ani_Liceu, StagiuMilitar}

Prin comparaţie cu „relaţia-mamut", economia de timp este semnificativă la obţinerea datelor privitoare la examene, prin joncţiunea cu relaţia NOTE. Risipa de timp şi efort se fac simţite la obţinerea informaţiilor extrase din STUD_DATEGENERALE şi STUD_LICEU, întrucât sunt necesare joncţiuni suplimentare cu tabela STUDENŢI. Plus timpul necesar gestionării restricţiei referenţiale.

Inclusă de mulţi autori la „capitolul" denormalizare, ruperea sau partiţionarea pe verticală adaugă redundanţă (apar restricţii referenţiale suplimentare), dar nu economiseşte din joncţiuni, ci dimpotrivă. Şi aici s-ar potrivi observaţia lui Chris Date, precum că mulţi autori cad în capcana confuziei dintre denormalizare şi redundanţă. Decizia segmentării pe verticală a bazei de date trebuie asumată numai dacă numărul de atribute din tabelă şi lungimea lor sunt foarte mari, volumul datelor poate creşte de o manieră impresionantă în timp şi, în plus, gradul de accesare a diferitelor atribute sau categorii de atribute prezintă diferenţe semnificative.

CAPITOLUL 10

Temporalitate şi acces în baze de date

Un criteriu esenţial în evaluarea calităţii unei baze de date este cel al timpului. Validitatea temporală, mai precis, lipsa de adecvare a schemei bazei la modificările ce apar ulterior în sistem reprezintă coşmarul multor administratori şi dezvoltatori de aplicaţii. Ideal ar fi ca în momentul proiectării schemei bazei să se cunoască direcţiile în care evoluează domeniul, legislaţia şi procedurile modulului/sistemului pentru care se dezvoltă aplicaţia, astfel încât structura să fie suficient de flexibilă pentru adaptări ulterioare. Echipele norocoase au câte un Brucan al lor sau, după caz, o Mama Omida. Celelalte trebuie să procedeze ca noi, adică să-şi bată capul identificând eventualele probleme, iminente sau mai îndepărtate. Şi, după cum vedea, problemele sunt mult mai aproape decât credem...

10.1. Validitatea temporală a unei scheme

Baza de datè dedicată vânzărilor, mai precis triada PRODUSE {CodPr, DenPr, UM, ProcTVA, Grupa}, FACT {NrFact, CodCl, DataFact, Obs} şi LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit} (vezi paragraful 5.2), este mai mult decât cunoscută, este celebră. Puţine sunt cărţile dedicate bazelor de date care să reziste tentaţiei de a nu da un exemplu de acest gen. Fireşte, autorii americani sunt „scutiţi" de TVA, chiar dacă le rămân suficient de multe alte taxe. Ei bine, pentru mediul economic european, şi mai ales pentru mediul de afaceri românesc, această schemă, celebră cum e ea, este inadecvată pe termen lung, ba chiar mediu!

De ce? Simplu! Valoarea oricărei facturi se calculează, în lipsa unor atribute redundante, cu binecunoscuta interogare:

```
SELECT SUM(cantitate * pretunit) AS ValFactFaraTVA,
   SUM(cantitate * pretunit * (1 * procTVA)) AS ValFactTotala
FROM fact f
   INNER JOIN linii_fact lf ON f.nrfact=lf.nrfact
   INNER JOIN produse p ON lf.codpr=p.codpr
WHERE f.nrfact = 1
```

TVA-ul, pentru fiecare linie din factură, se determină pornind de la ProcTVA din produse, dar acest atribut indică doar procentul *curent* al TVA pentru produsul

cu pricina. Să presupunem că un supermarket deschis în 1998 şi destul de bine automatizat (case de marcat, stocuri *online*) vinde, prin altele, şi cărţi. Procentul de TVA a cunoscut variaţii sensibile în ultimii ani. Atunci, dacă în luna februarie 2005 dorim să analizăm vânzările de cărţi pe ultimii cinci ani, ce relevanţă mai au valorile furnizate prin interogarea de mai sus?

Rezolvarea acestei probleme esenţiale pentru bunul mers al aplicaţiei presupune stocarea procentului TVA din momentul preluării facturii în baza de date. Prima soluţie ar ţine cont de modul în care are loc, la intervale mai mari sau mai mici, modificarea procentelor de TVA ale produselor – vezi figura 10.1.

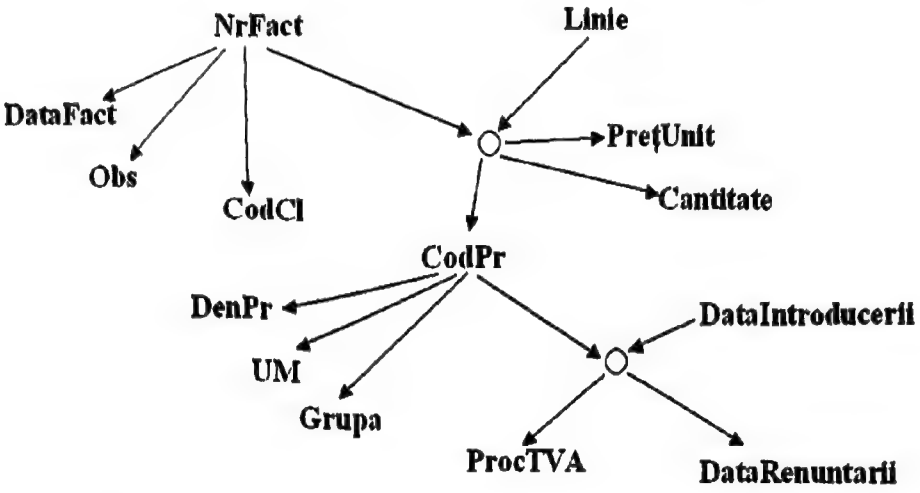


Figura 10.1. Procentul TVA pentru orice produs se poate schimba oricât de des

Pe baza acestui graf schema devine:

PRODUSE {<u>CodPr</u>, DenPr, UM, Grupa}
TVA_PRODUSE {<u>CodPr</u>, <u>DataIntroducerii</u>, DataRenunţarii, ProcTVA},

iar FACT şi LINII_FACT rămân neschimbate. Salvarea validităţii temporale se face cu pierderi grele de viteză, întrucât orice recompunere a valorii unei facturi este mult mai laborioasă (vezi figura 10.2):

```
SELECT f.NrFact, DataFact, Linie, DenPr,
     cantitate * pretunit AS ValFaraTVA,
     t.ProcTVA,
     cantitate * pretunit * (1 + t.procTVA/100) AS ValCuTVA
FROM fact f
   INNER JOIN linii_fact lf ON f.nrfact=lf.nrfact
   INNER JOIN produse p ON lf.CodPr=p.CodPr
   INNER JOIN tva_produse t ON lf.CodPr=t.CodPr AND
     f.datafact BETWEEN DataIntroducerii AND
       NVL(DataRenuntarii, SYSDATE)
ORDER BY f.NrFact
```

NRFACT	DATAFACT	LINIE	DENPR	VALFARATVA	PROCTVA	VALCUTVA
1	30-DEC-01	1	P1	1500000	22	1830000
1	30-DEC-01	2	P2	700000	19	833000
2	26-JUL-04	1	P3	780000	19	928200
3	30-DEC-01	2	P2	1200000	19	1523200
3	30-DEC-01	3	P3	1530000	19	1820700
4	26-JUL-04	2	P1	840000	19	999600
4	26-JUL-04	1	P1	1000000	19	1190000
5	31-JAN-99	1	P4	1710000	9	1863900
5	31-JAN-99	2	P5	754000	9	821860
6	12-DEC-03	1	P3	913500	19	1087065
7	10-JAN-05	3	P2	1425000	0	1425000
7	10-JAN-05	1	P6	1185000	19	1410150
7	10-JAN-05	2	P2	1530000	0	1530000
8	05-JAN-05	1	P1	1232500	9	1343425

Figura 10.2. Calculul corect al valorii produselor facturate

Pe de altă parte, şi *trigger*-ele de inserare/modificare/ştergere trebuie să „cotrobăie" prin TVA_PRODUSE, pentru a determina „procentul corect" în vederea actualizărilor corespunzătoare ale atributelor calculate din FACT: `ValTotală` şi `TVAFact`. *Listing*-ul de creare a noii tabele şi de actualizare a PRODUSE, FACT şi LINII_FACT (*listing* 10.1) este disponibil pe Web.

O parte dintre dvs. s-ar gândi ca, în loc să operăm modificările de TVA la toate produsele pe care le comercializăm, mai bine definim grupe de TVA. De exemplu, definim grupa *cărţi*, iar o revenire la cota de 9% s-ar traduce prin modificarea (şi jurnalizarea) procentului numai la această grupă. Schema ar fi:

GRUPE_TVA {IdGrupaTVA, DenGrupaTVA, Observaţii}
PRODUSE {CodPr, DenPr, UM, Grupa, IdGrupaTVA}
TVA_Grupe {IdGrupaTVA, DataIntroducerii, DataRenunţarii, ProcTVA}

Avantajul ar fi economia de timp la modificări repetate ale cotei de TVA ale diferitelor categorii de produse. Există, însă, şi două dezavantaje majore. Unul ar fi că mai apare o relaţie/restricţie referenţială. Iar, în al doilea şi cel mai important rând, soluţia este compromisă dacă un produs este mutat, în timp, dintr-o categorie de TVA în alta. Nu ştiu dacă e cel mai nimerit exemplu (sigur, nu cel mai moral cu putinţă), dar ar fi posibil ca anumite categorii de publicaţii să fie considerate cam deocheate şi, mergând pe ideea că totul (mai ales plăcerea) se plăteşte, legiuitorul român stabileşte un procent de TVA diferit pentru o parte dintre cărţi/reviste.

A treia variantă ar fi una dintre cele evocate în capitolul 9 – folosirea de atribute redundante, sintetice. Astfel, în tabela PRODUSE păstrăm atributul `ProcTVA`, a cărui valoare indică procentul curent de TVA aplicat la vânzarea produsului, iar în tabela LINII_FACT introducem atributul `TVALinie`:

PRODUSE {CodPr, DenPr, UM, Grupa, ProcCurentTVA}
TVA_PRODUSE {CodPr, DataIntroducerii, DataRenunţarii, ProcTVA}
LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit, TVALinie}

Deşi redundantă în bună măsură, structura este una acceptabilă şi „rezistentă" în timp. Fireşte, se poate merge şi mai departe cu discuţia pentru această bază de date. Ţine de regulile unei economii libere ca orice firmă să anunţe perioade de reduceri de preţ, fie pentru scăpa de stocuri, fie pentru a impulsiona vânzările la un sortiment nou, necunoscut. Pe perioada promoţională a unui produs, preţul său „normal" de vânzare este `PreţPromoţional`, în timp ce în lipsa promoţiei preţul de vânzare este `PreţCurent`. Cu toate acestea, graful DF din figura 10.3 mai conţine şi atributul `PreţUnitar`, şi aceasta deoarece atunci când un client cumpără o cantitate mai mare sau ne oferă nişte condiţii superavantajoase (sau are cunoştinţe în parlament/guvern), anumite produse pot fi vândute sub nivelul preţului curent.

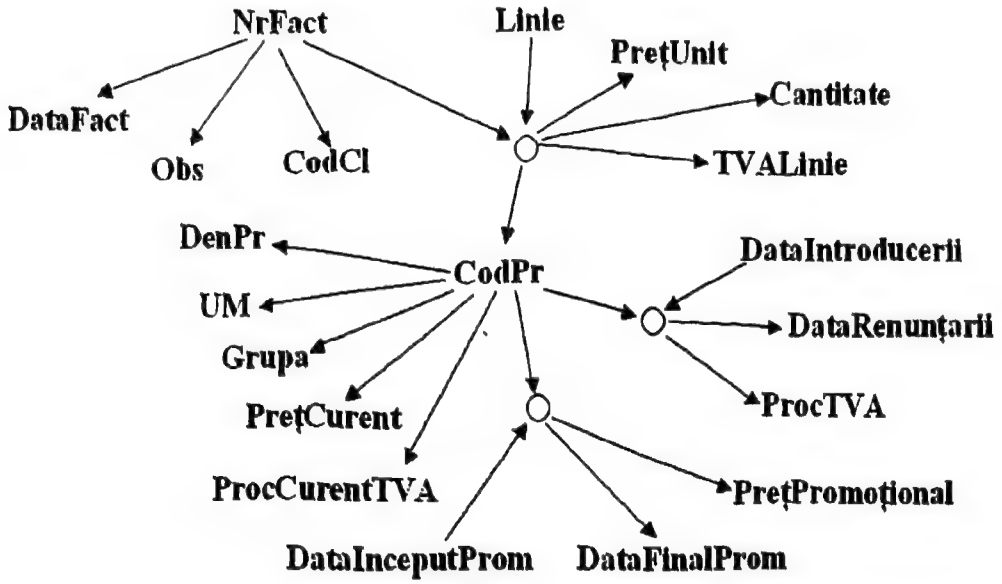


Figura 10.3. Preţuri şi campanii promoţionale

Schema rezultată din graf ar fi:

PRODUSE {`CodPr`, `DenPr`, `UM`, `Grupa`, `PreţCurent`, `ProcCurentTVA`}
TVA_PRODUSE {`CodPr`, `DataIntroducerii`, `DataRenunţarii`, `ProcTVA`}
PROMOŢII {`CodPr`, `DataÎnceputProm`, `DataFinalProm`, `PreţPromoţional`}
FACT {`NrFact`, `CodCl`, `DataFact`, `Obs`}
LINII_FACT {`NrFact`, `Linie`, `CodPr`, `Cantitate`, `PreţUnit`, `TVALinie`}

Listing-ul 10.2 s-a prelungit peste ceea ce era de aşteptat, deoarece, faţă de versiunea anterioară a tabelei PRODUSE, atributul `ProcTVA` este „rebotezat" `ProcCurentTVA`, iar în modificarea trebuie să se propage în toate blocurile PL/SQL care fac apel la acest atribut.

Listing 10.2. Modificări ale schemei, ale unor declanşatoare şi noi funcţii pentru determinarea valorii implicite a preţului de vânzare

```
ALTER TABLE produse RENAME COLUMN ProcTVA TO ProcCurentTVA ;

ALTER TABLE produse ADD PretCurent NUMBER(12) ;

UPDATE produse SET PretCurent=1500 WHERE CodPr=1 ;
UPDATE produse SET PretCurent=1400 WHERE CodPr=2 ;
UPDATE produse SET PretCurent=1300 WHERE CodPr=3 ;
UPDATE produse SET PretCurent=1200 WHERE CodPr=4 ;
UPDATE produse SET PretCurent=1450 WHERE CodPr=5 ;
UPDATE produse SET PretCurent=1500 WHERE CodPr=6 ;

DROP TABLE promotii ;
CREATE TABLE promotii (
    CodPr NUMBER(6) NOT NULL REFERENCES produse (CodPr),
    DataInceputProm DATE,
    DataFinalProm DATE,
    PretPromotional NUMBER(12)
    ) ;

INSERT INTO promotii VALUES (1, DATE'2005-01-01', NULL, 1475) ;
INSERT INTO promotii VALUES (3, DATE'2005-10-20', DATE'2005-02-02', 1325) ;
INSERT INTO promotii VALUES (5, DATE'2005-02-01', NULL, 1325) ;

-- NOUL declanşator de stergere in LINII_FACT - la nivel de LINIE
CREATE OR REPLACE TRIGGER trg_lf_del
    BEFORE DELETE ON linii_fact FOR EACH ROW
BEGIN
    pac_vanzari.v_trg_liniifact := TRUE ;
    pac_vanzari.v_nrfact := :OLD.NrFact ;
    pac_vanzari.v_liniestearsa := :OLD.Linie ;
    UPDATE fact SET
        valtotala = valtotala - :OLD.cantitate *
            :OLD.pretunit * (1 +
             (SELECT ProcCurentTVA FROM produse WHERE codpr=:OLD.codpr)),
        TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
            (SELECT ProcCurentTVA FROM produse WHERE codpr=:OLD.codpr)
    WHERE nrfact=:OLD.nrfact ;
    pac_vanzari.v_trg_liniifact := FALSE ;
END ;
/

DELETE FROM linii_fact WHERE NrFact >= 10 ;
DELETE FROM fact WHERE NrFact >= 10 ;

INSERT INTO fact (nrfact, datafact, codcl) VALUES (10, SYSDATE,1001);
INSERT INTO fact (nrfact, datafact, codcl) VALUES (11, SYSDATE,1001);
INSERT INTO fact (nrfact, datafact, codcl) VALUES (12, SYSDATE,1002);
INSERT INTO fact (nrfact, datafact, codcl) VALUES (13, SYSDATE,1003);
```

```
INSERT INTO fact (nrfact, datafact, codcl) VALUES (14, SYSDATE,1002);

-------------------------------------------------------
CREATE OR REPLACE FUNCTION f_datafact (nrfact_ fact.NrFact%TYPE)
   RETURN DATE
IS
   v_data DATE ;
BEGIN
   SELECT DataFact INTO v_data FROM fact
   WHERE NrFact=nrfact_ ;
   RETURN v_data ;
END f_datafact ;
/

-------------------------------------------------------
CREATE OR REPLACE FUNCTION f_pret (codpr_ produse.codpr%TYPE,
   nrfact_ fact.nrfact%TYPE)
      RETURN produse.PretCurent%TYPE
IS
   v_pret produse.PretCurent%TYPE ;
BEGIN
   SELECT PretPromotional INTO v_pret FROM promotii
   WHERE CodPr = codpr_ AND f_datafact (nrfact_) BETWEEN
      DataInceputProm    AND NVL(DataFinalProm,SYSDATE) ;
   RETURN v_pret ;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      SELECT PretCurent INTO v_pret FROM produse
         WHERE CodPr=codpr_ ;
      RETURN v_pret ;
END ;
/

-- modificam declansatorul BEFORE-INSERT-ROW in LINII_FACT
--  (versiunea anterioara era in listing-ul 9.6 (partea a II-a)
CREATE OR REPLACE TRIGGER trg_lf_ins0
    BEFORE INSERT ON linii_fact FOR EACH ROW
BEGIN
   pac_vanzari.v_trg_liniifact := TRUE ;
   :NEW.Linie := pac_vanzari.f_nrlinii (:NEW.NrFact) + 1 ;
   UPDATE fact SET NrLinii = NrLinii + 1
      WHERE NrFact = :NEW.NrFact ;
   :NEW.PretUnit := f_pret (:NEW.CodPr, :NEW.NrFact) ;
   pac_vanzari.v_trg_liniifact := FALSE ;
END ;
/

-----------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_ins
AFTER INSERT ON linii_fact FOR EACH ROW
BEGIN
   pac_vanzari.v_trg_liniifact := TRUE ;
```

```
    UPDATE fact SET
      valtotala = NVL(valtotala,0) +
        :NEW.cantitate *:NEW.pretunit * (1 +
        (SELECT ProcCurentTVA FROM produse WHERE codpr=:NEW.codpr)),
      TVAfact = NVL(TVAfact,0) + :NEW.cantitate * :NEW.pretunit *
        (SELECT ProcCurentTVA FROM produse WHERE codpr=:NEW.codpr)
    WHERE nrfact=:NEW.nrfact ;
    pac_vanzari.v_trg_liniifact := FALSE ;
END ;


INSERT INTO linii_fact VALUES (10, 1, 4, 950, NULL) ;
INSERT INTO linii_fact VALUES (11, 2, 5, 580, NULL) ;
INSERT INTO linii_fact VALUES (11, 1, 3, 630, NULL) ;
INSERT INTO linii_fact VALUES (12, 1, 6, 790, NULL) ;
INSERT INTO linii_fact VALUES (12, 2, 2, 850, NULL) ;
INSERT INTO linii_fact VALUES (12, 3, 2, 950, NULL) ;
INSERT INTO linii_fact VALUES (13, 1, 1, 850, NULL) ;
```

Cele câteva date de test, precum şi rezultatele interogărilor din figura 10.4 ne dau garanţia că sistemul este funcţional.

```
SQL> SELECT * FROM PRODUSE ;

     CODPR DENPR           UM   GRUPA    PROCCURENTTVA PRETCURENT
---------- -------------- ---- -------- ------------- ----------
         1 P1             kg                    .19       1500
         2 P2             kg                    .09       1400
         3 P3             kg                    .19       1300
         4 P4             mc   Textile          .19       1200
         5 P5             ml   Cafea            .19       1450
         6 P6             buc  Bere             .19       1500

6 rows selected.

SQL> SELECT * FROM PROMOTII;

     CODPR DATAINCEP DATAFINAL PRETPROMOTIONAL
---------- --------- --------- ---------------
         1 01-JAN-05                     1475
         3 20-OCT-05 02-FEB-05           1325
         5 01-FEB-05                     1325

SQL> SELECT f.NrFact, f.DataFact,Linie, CodPr, Cantitate, PretUnit
  2  FROM fact f INNER JOIN linii_fact lf ON f.NrFact=lf.NrFact
  3  WHERE f.NrFact >= 8
  4  /

    NRFACT DATAFACT       LINIE      CODPR  CANTITATE   PRETUNIT
---------- --------- ---------- ---------- ---------- ----------
         8 05-JAN-05          1          1        850       1450
        10 03-FEB-05          1          4        950       1200
        11 03-FEB-05          1          5        580       1325
        11 03-FEB-05          2          3        630       1300
        12 03-FEB-05          1          6        790       1500
        12 03-FEB-05          2          2        850       1400
        12 03-FEB-05          3          2        950       1400
        13 03-FEB-05          1          1        850       1475
```

Figura 10.4. Calculul valorii implicite a preţului de vânzare

Şi cum acest capitol va fi presărat cu întrebări „istorice", să ne gândim la problema : putem şti, pentru ultimii şapte ani, care au fost preţurile unitare „curente" ale produsului X? Pentru a răspunde corect, ar trebui să lămurim dacă întrebarea are rost sau nu. Credem că, în cazul acesta, nu prea! Totuşi, celor insistenţi le-am putea pune la dispoziţie toate preţurile la care au fost vândute produsele pe baza înregistrărilor stocate în LINII_FACT. Aici sunt consemnate însă şi preţurile mai mici datorate reducerilor făcute clienţilor fideli sau care au cumpărat cantităţi foarte mari din acest produs, iar, pe de altă parte, s-ar putea ca, dacă produsul este foarte scump şi vânzările foarte rare, o parte dintre preţuri să nu fi fost stocate. Pentru această a două speţă, să luăm un exemplu. La un supermagazin există un televizor cu cristale lichide şi toate „bunătăţurile" la un preţ exorbitant de 170 miloane de lei. După şase luni, magazinul mai lasă din preţ - 160 milioane de lei, dar degeaba! După alte două reduceri, la 155 şi 147 milioane, urmează o depreciere severă a leului faţă de euro, iar preţul ajunge la 159 milioane. Abia după o altă ieftinire, la 153 de milioane, produsul este vândut. Ei bine, din toată această evoluţie a preţului, în schema propusă va fi preluată doar ultima valoare.

10.2. Consemnarea evoluţiei unor parametri. „Refacerea" unei valori anterioare

Pentru aplicaţiile tranzacţionale, consemnarea operaţiunilor în momentul producerii lor sau nu mult timp după aceea este rezolvată în cea mai mare parte a cazurilor prin schema de stocare. Astfel, pentru baza de date VÂNZĂRI, preluarea în baza de date a vânzărilor şi încasărilor, deşi uneori cu un uşor decalaj temporal (mai ales la încasări), reflectă totuşi ordinea cronologică a evenimentelor. Analog stau lucrurile şi pentru centrul de închirieri din capitolul 7, pentru firma de transport călători din capitolul 8, „bancomatele" din capitolul 9 sau miniaplicaţia de contabilitate generală tratată în ultimele trei capitole.

Din raţiuni de simplificare şi, de cele mai multe ori, datorită lipsei de relevanţă pentru firmă, evoluţia unor parametri nu este consemnată în bază. Bunăoară, în baza de date VÂNZĂRI am fost foarte atenţi, mai ales în acest capitol, cu evoluţia procentului TVA şi a preţului fiecărui produs, însă faptul că un client îşi poate schimba, în timp, sediul sau numărul de telefon nu ne-a preocupat absolut deloc. La urma urmei, este chiar aşa de important să ştim pe ce stradă îşi avea clientul X sediul în decembrie 2000 (presupunând că aplicaţia noastră e implementată în iunie 1997)? Ei bine, există firme şi organizaţii, precum ROMTELECOM, DISTRIGAZ, S.C. ELECTRICA, bănci, şi mai ales Poliţia şi serviciul de evidenţa populaţiei pentru care cunoaşterea pe termen lung a tuturor datelor legate de domiciliul clienţilor are o mare importanţă. Pentru aceste cazuri, graful DF poate fi cel din figura 10.5.

`DataInițială` şi `DataFinală` reprezintă intervalul pentru care este valabilă acea componentă a adresei. Valoarea NULL a atributului `DataFinală` ar semnala faptul că respectiva componentă a adresei este valabilă în prezent. Codul, denumirea şi codul fiscal sunt proprietăţile cele mai constante ale clientului, aşa că modificarea

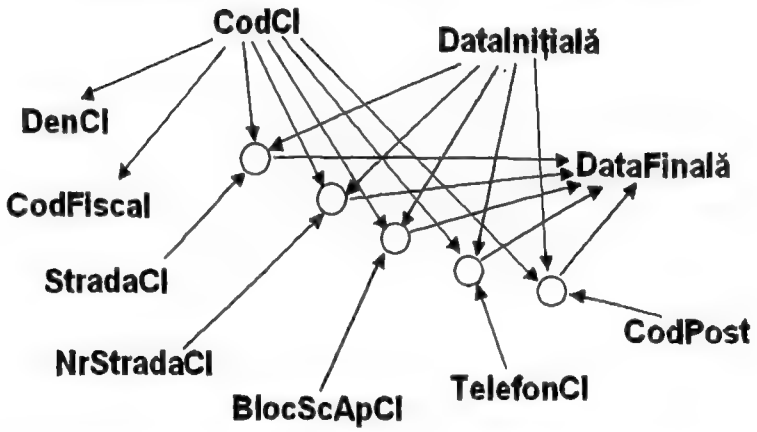


Figura 10.5. Preluarea oricărei modificări de adresă

lor este improbabilă. În sursele dependenţelor compuse a fost inclus şi atributul `DataIniţială`, deoarece, în timp, un client poate reveni la o adresă anterioară sau în apropierea unei adrese anterioare.

Structura se complică rău de tot:

CLIENŢI {CodCl, DenCl, CodFiscal}
CLIENŢI_STRĂZI {CodCl, StradaCl, DataInițială, DataFinală}
CLIENŢI_NR {CodCl, NrStradaCl, DataInițială, DataFinală}
CLIENŢI_BLOC {CodCl, BlocScApCl, DataInițială, DataFinală}
CLIENŢI_TEL {CodCl, TelefonCl, DataInițială, DataFinală}
CLIENŢI_CODPOST {CodCl, CodPost, DataInițială, DataFinală}

Dacă ţinem cont că majoritatea interogărilor legate de clienţi privesc adresa curentă a acestora, n-ar strica totuşi să aplicăm ceea ce ne-a predicat capitolul 9: că o anumită doză de redundanţă nu strică pe la casa omului:

CLIENŢI2 {CodCl, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost}

Listing-ul de creare a celor cinci tabele este disponibil pe Web (*listing*-ul 10.3). Probabil că cei mai mulţi dintre detractorii schemei stufoase la care am ajuns de dragul unei banale adrese, mai precis, a istoricului său, vor acuza complexitatea presupusă de gestiunea celor şase tabele. Ei bine, lucrurile nu sunt chiar aşa. Putem ascunde multe dintre detalii lucrând „în fundal" cu declanşatoarele. Astfel, orice modificare a vreunuia dintre atributele din adresă poate fi imortalizată în maniera prezentată în *listing*-ul 10.4.

Listing 10.4. O funcție și un declanșator pentru „contabilizarea" modificărilor adresei unui client

```
CREATE OR REPLACE FUNCTION f_data_ultimei_modif (
    codcl_ clienti2.CodCl%TYPE,
    atribut_ VARCHAR2) RETURN DATE
AS
    v_data DATE ;
    v_tabela VARCHAR2(20) ;
BEGIN
    CASE atribut_
    WHEN 'StradaCl' THEN
        v_tabela := 'CLIENTI_STRAZI';
    WHEN 'NrStradaCl' THEN
        v_tabela := 'CLIENTI_NR';
    WHEN 'BlocScApCl' THEN
        v_tabela := 'CLIENTI_BLOC';
    WHEN 'Telefon' THEN
        v_tabela := 'CLIENTI_TEL';
    WHEN 'CodPost' THEN
        v_tabela := 'CLIENTI_CODPOST';
    END CASE ;

    EXECUTE IMMEDIATE 'SELECT MAX(DataFinala)  FROM '
        || v_tabela || ' WHERE CodCl = :1 ' INTO v_data USING codcl_ ;
    RETURN v_data ;
END ;
/

-------------------------------------------------------------------------------
-- modificarea unui element din adresa in tabela CLIENTI2
CREATE OR REPLACE TRIGGER trg_clienti_upd2
    AFTER UPDATE OF StradaCl, NrStradaCl, BlocScApCl,
    Telefon, CodPost ON clienti2 FOR EACH ROW
BEGIN
    -- s-a modificat strada ?
    IF NVL(:NEW.StradaCl,' ') <> NVL(:OLD.StradaCl,' ') THEN
        INSERT INTO clienti_strazi VALUES (:OLD.CodCl, NVL(:OLD.StradaCl,' '),
        NVL(f_data_ultimei_modif (:OLD.CodCl, 'StradaCl'), DATE'1990-01-01'),
        SYSDATE);
    END IF ;

    -- s-a modificat numarul ?
    IF NVL(:NEW.NrStradaCl,' ') <> NVL(:OLD.NrStradaCl,' ') THEN
        INSERT INTO clienti_nr VALUES (:OLD.CodCl,NVL(:OLD.NrStradaCl,' '),
        NVL(f_data_ultimei_modif (:OLD.CodCl, 'NrStradaCl'), DATE'1990-01-01'),
        SYSDATE);
    END IF ;

    -- s-a modificat blocul/scara/apartamentul ?
    IF NVL(:NEW.BlocScApCl,' ') <> NVL(:OLD.BlocScApCl,' ') THEN
        INSERT INTO clienti_bloc VALUES (:OLD.CodCl,NVL(:OLD.BlocScApCl,' ') ,
```

```
      NVL(f_data_ultimei_modif (:OLD.CodCl, 'BlocScApCl'), DATE'1990-01-01'),
      SYSDATE);
   END IF ;

   -- s-a modificat numarul de telefon ?
   IF NVL(:NEW.Telefon, ' ') <> NVL(:OLD.Telefon,' ') THEN
      INSERT INTO clienti_tel VALUES (:OLD.CodCl,NVL(:OLD.Telefon, ' '),
      NVL(f_data_ultimei_modif (:OLD.CodCl, 'Telefon'), DATE'1990-01-01'),
      SYSDATE);
   END IF ;

   -- s-a modificat codul postal ?
   IF NVL(:NEW.CodPost,0) <> NVL(:OLD.CodPost,0) THEN
      INSERT INTO clienti_codpost VALUES (:OLD.CodCl, NVL(:OLD.CodPost,0),
      NVL(f_data_ultimei_modif (:OLD.CodCl, 'CodPost'), DATE'1990-01-01'),
      SYSDATE);
   END IF ;
END ;
/
```

Funcţia determină care a fost data ultimei modificări, pentru clientul-parametru, a străzii, numărului... Dacă este vorba de prima modificare a unui client/atribut, atunci data returnată este 1 ianuarie 1990, deşi mai corectă ar fi fost valoarea de implementare a aplicaţiei - 1 ianuarie 1997. Pentru a ne mai dezmorţi un pic oasele în PL/SQL, am folosit un mic artificiu - SQL dinamic, astfel încât să nu scriem câte un `SELECT` fiecăreia dintre cele cinci tabele. Declanşatorului nu-i mai rămâne decât să verifice care dintre atribute s-a modificat şi să facă inserarea în tabela „istoric" corespunzătoare.

Cu această schemă, aflarea străzilor pe care îşi aveau clienţii sediile la momentul 8 februarie 2005, ora 9:42 nu ridică probleme deosebite. O interogare s-ar putea baza pe o facilitate mai recentă din dialectul SQL/Oracle, şi anume posibilitatea de a „boteza" rezultatul unei interogări (STRĂZI) şi a-l folosi în fraza `SELECT` următoare ca o tabelă în toată regula:

```
WITH strazi AS
   (SELECT codcl, stradacl, datainitiala,
     datafinala - 1 /(24*60*60) AS datafinala
    FROM clienti_strazi
    UNION
    SELECT c,codcl, stradacl,
     NVL((SELECT MAX(datafinala) FROM clienti_strazi
       WHERE codcl=c.codcl)
     ,TO_DATE('1990-01-01','YYYY-MM-DD')),
     SYSDATE + 1 / (24*60*60)
   FROM clienti2 c
   ORDER BY codcl, datainitiala
```

```
  )
SELECT dencl, s.stradacl
FROM clienti2 c INNER JOIN strazi s ON c.codcl=s.codcl AND
  TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
  BETWEEN datainitiala AND datafinala
```

Pentru ca data iniţială a unui interval să nu fie identică cu cea finală a intervalului precedent, `datafinală` a fost decrementată cu o secundă, adică 1 / (24*60*60) dintr-o zi (24 de ore, 60 de minute şi 60 de secunde). Deşi nu parcurgem o lucrare dedicată SQL-ului, vă mai propun o soluţie:

```
SELECT dencl, c.codcl,
  NVL((SELECT stradacl FROM clienti_strazi
    WHERE codcl=c.codcl AND
    TO_DATE('2005-02-08 09:42:00',
    'YYYY-MM-DD HH24:MI:SS') BETWEEN
    datainitiala AND datafinala-1/(24*60*60)),
  c.stradacl) stradacl
FROM clienti2 c
```

Pe calapodul acestei secunde soluţii suntem în măsură să furnizăm un răspuns rezonabil la întrebarea: *care era adresa completă a clientului CL1 la 8 februarie 2005, ora 9:42:00 („ora bazei de date")?* Cele trei interogări sunt salvate în *listing*-ul 10.5 de pe Web.

```
SELECT dencl, c.codcl,
  NVL((SELECT stradacl FROM clienti_strazi
  WHERE codcl=c.codcl AND
  TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
  BETWEEN datainitiala AND datafinala-1/(24*60*60)),
  c.stradacl) stradacl,
  NVL((SELECT nrstradacl FROM clienti_nr
  WHERE codcl=c.codcl AND
  TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
  BETWEEN datainitiala AND datafinala-1/(24*60*60)),
  c.nrstradacl) nrstradacl,
  NVL((SELECT blocscapcl FROM clienti_bloc
  WHERE codcl=c.codcl AND
  TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
  BETWEEN datainitiala AND datafinala-1/(24*60*60)),
  c.blocscapcl) blocscapcl,
  NVL((SELECT telefon FROM clienti_tel
  WHERE codcl=c.codcl AND
  TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
```

```
    BETWEEN datainitiala AND datafinala-1/(24*60*60)),
    c.telefon) telefon,
    NVL((SELECT codpost FROM clienti_codpost
    WHERE codcl=c.codcl AND
    TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
    BETWEEN datainitiala AND datafinala-1/(24*60*60)),
    c.codpost) codpost
FROM clienti2 c
WHERE dencl='CL1'
```

Urmărirea evoluţiei unor parametri poate avea o valoare mai mult decât pur informativă, fiind uneori chiar vitală pentru contabilitatea şi finanţele unei firme. În acest sens, continuăm cu un exemplu care a dat ceva bătăi de cap proiectanţilor de baze de date: gestiunea stocurilor. Intrările în gestiune ale materiilor prime, materialelor şi celorlalte elemente patrimoniale care se încadrează la categoria *stocuri* sunt contabilizate la preţul de cumpărare (dacă au fost cumpărate), preţul de producţie (dacă au fost produse *in-house*) ş.a.m.d. Stocurile sunt destinate consumului, adică producerii altor sortimente, sau vânzării[1]. Problema este că, indiferent de tipul ieşirii din gestiune, evaluarea stocurilor la ieşire se face după una dintre metodele: FIFO – *First In, First Out*, adică *primul intrat, primul ieşit*, LIFO – *Last In, Last Out*, adică *ultimul intrat, primul ieşit*, sau CMP – *cost mediu ponderat* (aici iniţialele sunt „româneşti").

Să presupunem că sortimentul X este cumpărat (de fapt, i se face recepţia) pentru prima dată pe 2 februarie 2005, ora 10:00, prilej cu care se întocmeşte o notă de recepţie, iar intrarea „primeşte" identificatorul 12345. Cantitatea intrată este 5, iar preţul unitar 1000 de lei (grei). Pe 3 februarie, ora 9:00, se recepţionează încă 3 unităţi din acelaşi sortiment, la preţul unitar de 1100 lei:

IdIntrare	CantIntrată	PreţUnitIntrare	ValoareIntrare	CantRămasă
12345	5	1000	5000	5
12467	3	1100	3300	3

Vorbim, practic, de două tranşe ale acestui produs, una la preţul unitar de 1000 de lei şi o alta la 1100. După cele două intrări, are loc darea în consum a 6 unităţi din sortiment[2]. Întrucât firma aplică principiul FIFO, „descărcarea" va începe cu prima tranşă. Cum însă cantitatea ieşită este 6, iar prima tranşă are numai 5 unităţi, „se va muşca" o unitate şi din a doua tranşă:

IdIeşire	CantIeşire	ValoareIeşire	IdIntrare	CantDescărcată
14101	6	6100	12345	5
			12467	1

1. Trecem sub tăcere alte tipuri de operaţiuni cu stocurile rezultate din inventarieri, degradări, calamităţi etc.
2. Documentul care se întocmeşte este, de obicei, bonul de consum

După această ieşire, situaţia celor două tranşe este următoarea:

IdIntrare	CantIntrată	PreţUnitIntrare	ValoareIntrare	CantRămasă
12345	5	1000	5000	0
12467	3	1100	3300	2

Schema bazei trebuie să consemneze nu numai intrările şi ieşirile de stocuri, ci şi corespondenţa ieşiri-intrări, astfel încât principiul FIFO să fie respectat. Mai mult, la un eventual control trebuie demonstrat că, la orice ieşire a materialului, descărcarea gestiunii s-a făcut la preţul[3] corect. În aceste condiţii, graful dependenţelor funcţionale ar putea fi, în linii mari, similar celui din figura 10.6.

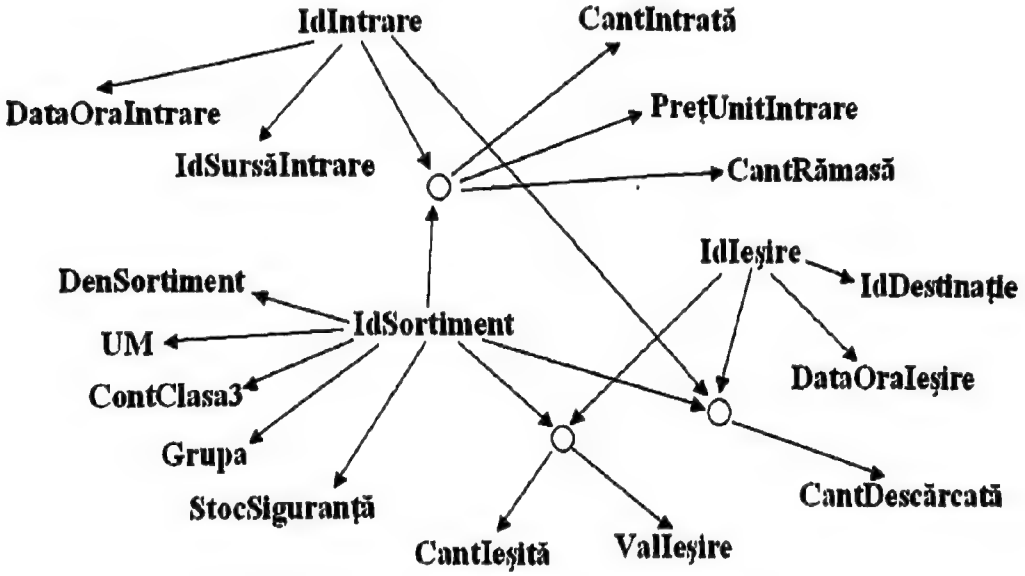


Figura 10.6. Gestiunea stocurilor prin metoda FIFO

Interesant e că tot acest mecanism de punere în corespondenţă a ieşirilor cu intrările de stocuri poate fi gestionat automat, la nivelul „logicii" bazei de date, firește, prin intermediul declanşatoarelor. *Listing*-ul 10.5 conţine comenzile de creare a tabelelor desprinse din graful din figura 10.6 şi câteva comenzi de populare, în consonanţă cu exemplul de mai sus.

Listing 10.5. Implementarea metodei FIFO prin declanşator

```
--
DROP TABLE iesiri_intrari ;
DROP TABLE iesiri_materiale ;
DROP TABLE iesiri ;
DROP TABLE intrari_materiale ;
DROP TABLE intrari ;
DROP TABLE materiale ;
```

3. Pentru firmă, preţul de descărcare la darea în consum sau vânzare este, de fapt, un cost, o cheltuială.

```
CREATE TABLE materiale (
    IdSortiment NUMBER(7) PRIMARY KEY,
    DenSortiment VARCHAR2(30) UNIQUE,
    UM VARCHAR2(10),
    ContClasa3 VARCHAR2(15),
    Grupa VARCHAR2(30),
    StocSiguranta NUMBER(10) DEFAULT 0 NOT NULL
    ) ;

CREATE TABLE intrari (
    IdIntrare NUMBER(10) NOT NULL PRIMARY KEY,
    DataOraIntrare DATE,
    IdSursaIntrare NUMBER(6)
    ) ;

CREATE TABLE intrari_materiale (
    IdIntrare NUMBER(10) NOT NULL REFERENCES intrari (IdIntrare),
    IdSortiment NUMBER(7) NOT NULL REFERENCES materiale (IdSortiment),
    CantIntrata NUMBER(10),
    PretUnitIntrare NUMBER(10,2),
    CantRamasa NUMBER(10) DEFAULT 0,
    PRIMARY KEY (IdIntrare, IdSortiment)
    ) ;

CREATE TABLE iesiri (
    IdIesire NUMBER(10) NOT NULL PRIMARY KEY,
    DataOraIesire DATE,
    IdDestinatie NUMBER(6)
    ) ;

CREATE TABLE iesiri_materiale (
    IdIesire NUMBER(10) NOT NULL REFERENCES iesiri (IdIesire),
    IdSortiment NUMBER(7) NOT NULL REFERENCES materiale (IdSortiment),
    CantIesita NUMBER(10),
    ValIesire NUMBER(12,2),
    PRIMARY KEY (IdIesire, IdSortiment)
    ) ;

CREATE TABLE iesiri_intrari (
    IdIesire NUMBER(10) NOT NULL REFERENCES iesiri (IdIesire),
    IdIntrare NUMBER(10) NOT NULL REFERENCES intrari (IdIntrare),
    IdSortiment NUMBER(7) NOT NULL REFERENCES materiale (IdSortiment),
    CantDescarcata NUMBER(10),
    PRIMARY KEY (IdIesire, IdIntrare, IdSortiment)
    ) ;

INSERT INTO materiale VALUES (1111, 'Sortiment X', 'kg', '3011.01', 'Materiale
constructie', 0) ;
INSERT INTO materiale VALUES (1112, 'Sortiment Y', 'kg', '3011.01', 'Materiale
constructie', 0) ;

INSERT INTO intrari VALUES
    (12345, TO_DATE('02/02/2005 10:00:00', 'DD/MM/YYYY HH24:MI:SS'), 123) ;
```

```
INSERT INTO intrari VALUES
   (12346, TO_DATE('02/02/2005 10:10:00', 'DD/MM/YYYY HH24:MI:SS'), 124) ;
INSERT INTO intrari VALUES
   (12467, TO_DATE('03/02/2005 09:00:00', 'DD/MM/YYYY HH24:MI:SS'), 123) ;

INSERT INTO intrari_materiale VALUES (12345, 1111, 5, 1000, 5) ;
INSERT INTO intrari_materiale VALUES (12345, 1112, 10, 1200, 10) ;
INSERT INTO intrari_materiale VALUES (12467, 1111, 3, 1100, 3) ;

INSERT INTO iesiri VALUES
   (14101, TO_DATE('03/02/2005 10:00:00', 'DD/MM/YYYY HH24:MI:SS'), 898) ;
INSERT INTO iesiri VALUES
   (14102, TO_DATE('03/02/2005 10:07:00', 'DD/MM/YYYY HH24:MI:SS'), 780) ;

CREATE OR REPLACE TRIGGER trg_iesiri_materiale
   BEFORE INSERT ON iesiri_materiale FOR EACH ROW
DECLARE
   CURSOR c_stoc (idsortiment_ materiale.IdSortiment%TYPE) IS
      SELECT * FROM intrari_materiale
      WHERE IdSortiment = idsortiment_ AND CantRamasa > 0
      ORDER BY IdIntrare ;
   v_cant_de_repartizat iesiri_materiale.CantIesita%TYPE ;
BEGIN
   v_cant_de_repartizat := :NEW.CantIesita ;
   :NEW.ValIesire := 0 ;
   FOR rec_stoc IN c_stoc (:NEW.IdSortiment) LOOP
      IF rec_stoc.CantRamasa >= v_cant_de_repartizat THEN
         INSERT INTO iesiri_intrari VALUES (:NEW.IdIesire, rec_stoc.IdIntrare,
            :NEW.IdSortiment, v_cant_de_repartizat) ;
         UPDATE intrari_materiale SET CantRamasa =
         CantRamasa - v_cant_de_repartizat
         WHERE IdIntrare=rec_Stoc.IdIntrare AND IdSortiment=rec_stoc.IdSortiment ;
         :NEW.ValIesire := :NEW.ValIesire + rec_stoc.PretUnitIntrare * v_cant_de_repartizat ;
         v_cant_de_repartizat := 0 ;
         EXIT ;
      ELSE
         INSERT INTO iesiri_intrari VALUES (:NEW.IdIesire, rec_stoc.IdIntrare,
            :NEW.IdSortiment, rec_stoc.CantRamasa) ;
         UPDATE intrari_materiale SET CantRamasa = 0
         WHERE IdIntrare=rec_Stoc.IdIntrare AND IdSortiment=rec_stoc.IdSortiment ;
         v_cant_de_repartizat := v_cant_de_repartizat - rec_stoc.CantRamasa ;
         :NEW.ValIesire := :NEW.ValIesire + rec_stoc.PretUnitIntrare *
         rec_stoc.CantRamasa ;
      END IF ;
   END LOOP ;
   IF v_cant_de_repartizat > 0 THEN
      RAISE_APPLICATION_ERROR (-20881, 'Stoc insuficient!') ;
   END IF ;
END ;
/

INSERT INTO iesiri_materiale VALUES (14101, 1111, 6, 0) ;
```

Punctul-cheie al *listing*-ului este declanşatorul de inserare în IEŞIRI_MATERIALE. La introducerea unei înregistrări în această tabelă, cursorul C_STOC se populează cu toate tranşele de preţ ale sortimentului respectiv la care stocul curent (cantitatea rămasă) este mai mare ca zero. Parcurgerea înregistrărilor din cursor are loc până la momentul în care suma cantităţilor repartizate este egală cu `:NEW.CantIeşită` - cantitatea indicată în bonul de consum. La fiecare tranşă identificată, tranşă ce corespunde unei înregistrări din cursor, se decrementează valoarea variabilei `v_cant_de_repartizat`, iniţializată înainte de parcurgerea cursorului cu valoarea `:NEW.CantIeşită`. Dacă după ieşirea din structura repetitivă variabila nu are valoarea zero, înseamnă că stocul sortimentului a fost insuficient pentru onorarea cantităţii de pe bonul de consum şi, prin urmare, se va declanşa o eroare.

Un alt caz de calculare retrospectivă a unei valori poate fi articulat în jurul exemplului din paragraful 9.5, dedicat curselor de Formula 1, al cărui graf este reprezentat în figura 9.4. După cele discutate mai sus, putem reproşa schemei din paragraful respectiv lipsa de „retrospectivă", întrucât datele pot fi gestionate doar pentru sezonul curent. Introducem deci două atribute, `Sezon` şi `Etapă`, plus alte două, `PuncteCursă` şi `PunctePenalizare`, necesare preluării rezultatelor fiecărei curse. În aceste condiţii, `PunctePoziţie` indică doar punctajul actual pentru plasarea pe un loc anume la finalul unei curse, în timp ce ultimele două atribute introduse sunt esenţiale pentru istoricul aplicaţiei. Graful este acum cel din figura 10.7, iar scriptul de creare şi populare a tabelelor este conţinut în *listing*-ul 10.6 (disponibil pe Web).

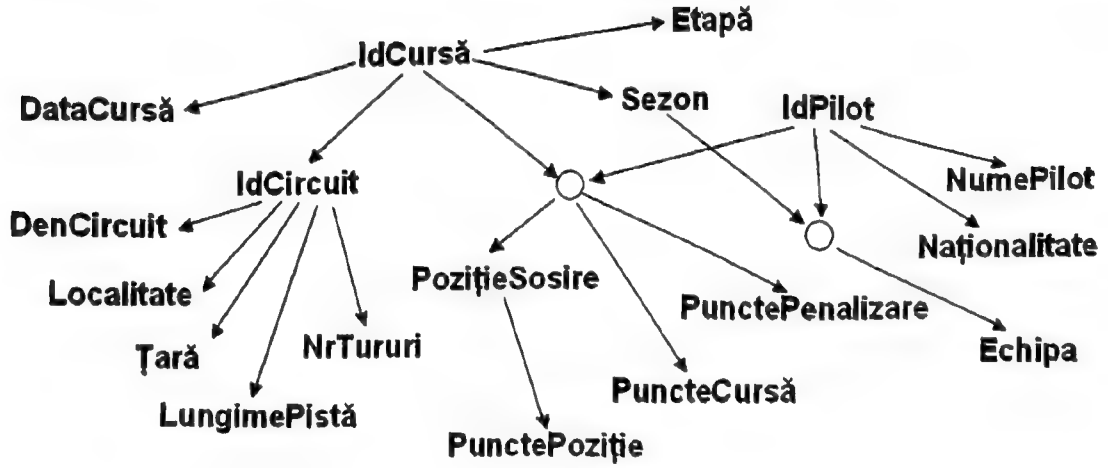


Figura 10.7. Rezultate – Formula 1

Să presupunem, nu fără realism, că o informaţie frecvent solicitată ar fi : *care era clasamentul piloţilor (sau echipelor) după etapa a doua a sezonului 2002 ?* La drept vorbind, această informaţie poate fi obţinută fără prea mare efort fără a opera nici o modificare în schema bazei de date - vezi interogarea şi rezultatul din figura 10.8. Astfel că recurgerea la denormalizare/postnormalizare, prin folosirea unei tabele redundante/sintetice de tipul CLASAMENTE_ETAPE (`Sezon`, `Etapă`, `IdPilot`, `PuncteCumulate`, `PenalizăriCumulate`) nu prea se justifică.

```
SQL> SELECT ROWNUM, t.*
  2  FROM
  3   (SELECT NumePilot, SUM(PuncteCursa) AS Puncte,
  4    SUM(PunctePenalizare) AS Penalizari,
  5     SUM(PuncteCursa - PunctePenalizare) AS Total
  6   FROM rezultate_curse rc INNER JOIN curse c ON rc.IdCursa=c.IdCursa
  7    INNER JOIN piloti p ON rc.IdPilot=p.IdPilot
  8   WHERE sezon=2002 AND Etapa <= 2
  9   GROUP BY NumePilot
 10   ORDER BY Total DESC) t
 11  /

    ROWNUM NUMEPILOT           PUNCTE PENALIZARI      TOTAL
---------- --------------- ---------- ---------- ----------
         1 Flash Hook              16          0         16
         2 Slash Gordon            20          8         12
         3 Nonaka John             11          0         11
         4 Benjamin Walter          9          0          9
         5 Yourdon Luke             6          0          6
```

Figura 10.8. Clasamentul după primele două etape ale sezonului 2002

Cu totul altfel stau lucrurile în cazul aplicaţiei de gestiune a competenţelor pe posturi şi persoane, descrisă în paragraful 8.2 (vezi graful din figura 8.8 şi schema din figura 8.9). Lucrurile sunt cu atât mai interesante cu cât atât posturile, cât şi competenţele şi angajaţii cunosc evoluţii semnificative în timp. Astfel, pentru compartimentul *Resurse umane*, important este nu numai nivelul actual al unei anumite competenţe pentru un angajat, dar şi modul în care a evoluat acest nivel în ultimii *n* ani. Puteam declara astfel ca specific acestei scheme faptul că interesează evoluţia a aproape tuturor atributelor din tabelele bazei, aşa că aplicarea schemei de lucru de la componentele adresei (figura 10.5) pare prea stufoasă.

Deşi laborioasă, o soluţie aplicabilă acestui gen de probleme ţine de crearea unei replici-jurnal pentru fiecare tabelă din bază, fără a schimba în vreun fel schema tabelelor existente. Ca exemplu, creăm tabela-jurnal pentru COMPETENŢE_PERSONAL, pe care o vom denumi IST_COMPETENŢE_PERSONAL:

```
DROP TABLE ist_competente_personal ;
CREATE TABLE ist_competente_personal (
 marca NUMBER (6) NOT NULL,
 idcompfrunza NUMBER (10) NOT NULL,
 nivel NUMBER(5) NOT NULL,
 data_init DATE DEFAULT CURRENT_DATE NOT NULL,
 data_fin DATE DEFAULT NULL,
 PRIMARY KEY (marca, idcompfrunza, data_init)
 ) ;
```

Cele două atribute suplimentare, `Data_Init` şi `Data_Fin`, vor indica perioada de valabilitate a înregistrării respective. Astfel, la inserarea unei linii în tabela-sursă se va face o inserare şi în „jurnalul" său, lucru destul de simplu de realizat prin declanşatorul operaţiunii:

```
CREATE OR REPLACE TRIGGER trg_competente_personal_ins
  AFTER INSERT ON competente_personal
  FOR EACH ROW
BEGIN
  INSERT INTO ist_competente_personal VALUES (:NEW.marca,
    :NEW.idcompfrunza, :NEW.Nivel, SYSDATE, NULL) ;
END ;
/
```

La ştergerea liniei din tabela-sursă, atributul `Data_Fin` va prelua data şi ora curentă, denulizarea sa semnalizând expirarea înregistrării:

```
CREATE OR REPLACE TRIGGER trg_competente_personal_del
  AFTER DELETE ON competente_personal
  FOR EACH ROW
BEGIN
  UPDATE ist_competente_personal SET data_fin = SYSDATE
  WHERE marca=:OLD.marca AND
 idcompfrunza=:OLD.IdCompFrunza AND data_fin IS NULL ;
END ;
/
```

Modificarea unei linii în tabela-sursă (tabela-stăpân) ar presupune declararea ca expirată a vechii copii a liniei şi inserarea noii instanţe. O variabilă ne va asigura că noua instanţă va fi valabilă la o secundă (1/(24*60*60)) de la expirarea precedentei instanţe.

```
CREATE OR REPLACE TRIGGER trg_competente_personal_upd
  AFTER UPDATE ON competente_personal
  FOR EACH ROW
DECLARE
  v_data DATE := SYSDATE ;
BEGIN
  -- se declara "expirate" vechile valori
  UPDATE ist_competente_personal SET data_fin = v_data
  WHERE marca=:OLD.marca AND
 idcompfrunza=:OLD.IdCompFrunza AND data_fin IS NULL ;

  -- se insereaza noile valori
  INSERT INTO ist_competente_personal VALUES (:NEW.marca,
    :NEW.idcompfrunza, :NEW.Nivel,
v_data + 1 /(24*60*60), NULL) ;
END ;
/
```

Pentru corectitudinea soluţiei ar trebui să folosim variabile publice, modificate în cele trei declanşatoare de mai sus şi „controlate" în *trigger*-ele de inserare şi

modificare ale tabelei IST_COMPETENŢE_PERSONAL, astfel încât vom evita alterarea jurnalizării prin comenzi directe (altfel decât prin declanşatoarele tabelei COMPETENŢE_PERSONAL). Cele trei declanşatoare de mai sus sunt grupate pe Web în *listing*-ul 10.8.

Extragerea informaţiilor curente din bază presupune consultarea exclusivă a tabelelor stabilite încă din capitolul 8, în timp ce scormonirea prin arhive reclamă folosirea tabelelor-jurnal. Astfel, dacă se doreşte aflarea nivelului tuturor competenţelor la care era evaluată Vasilescu Georgeta la 16 februarie 2005, ora 9:42, interogarea ar putea avea forma:

```
SELECT dencompetenta, nivel
FROM
   (SELECT marca, numepren FROM ist_personal9
    WHERE numepren='Vasilescu Georgeta' AND
      TO_DATE('2005-02-16 09:42','YYYY-MM-DD HH24:MI')
      BETWEEN data_init AND NVL(data_fin, SYSDATE)) p
   INNER JOIN
   (SELECT * FROM ist_competente_personal
    WHERE TO_DATE('2005-02-16 09:42','YYYY-MM-DD HH24:MI')
      BETWEEN data_init AND NVL(data_fin, SYSDATE)) cp
   ON p.marca=cp.marca
   INNER JOIN
      (SELECT idcompetenta, dencompetenta
       FROM ist_competente
       WHERE TO_DATE('2005-02-16 09:42',
         'YYYY-MM-DD HH24:MI') BETWEEN data_init AND
         NVL(data_fin, SYSDATE)) c
   ON cp.idcompfrunza=c.idcompetenta
```

Înainte de a ne freca palmele mulţumiţi de isprava noastră, ar trebui să ne gândim dacă lucrurile sunt într-adevăr în regulă pentru toate situaţiile de actualizare a tabelei-sursă. Veştile nu sunt din cale-afară de încurajatoare. Mecanismul este corect doar pentru atributele neimplicate în restricţiile referenţiale. Ce se întâmplă dacă în tabela PERSONAL9 se modifică marca unui angajat care, prin mecanismul `UPDATE CASCADE`, determină actualizarea liniilor-copil din COMPETENŢE_PERSONAL? Declanşatorul nostru – `trg_competente_personal_upd` – va contabiliza conştiincios expirarea înregistrării referitoare la vechea marcă şi va insera o linie nouă care se va referi la noua valoare a mărcii. Anxietatea se leagă de răspunsul corect la întrebarea: *cum se poate şti, în timp, că linii diferite din tabela-jurnal, cu valori diferite ale atributului* `Marcă`*, se referă, de fapt, la una şi aceeaşi persoană ?*

Primul impuls ar fi să interzicem orice modificare a atributelor de tip cheie surogat sau chiar a tuturor atributelor părinte din schemă. Însă acest lucru nu este întotdeauna realist, mai ales dacă atributul părinte nu este surogat sau dacă situaţia este una cu totul specială (două firme fuzionează şi bazele lor de date la fel). Aşa că ne încercăm norocul cu o altă variantă, pezentată în *listing*-ul 10.9. Tabela-istoric va

avea încă două atribute - `IdJurnalizare_Init` şi `IdJurnalizare_Fin` -, prin care consemnăm tranzacţia de inserare, modificare sau ştergere din COMPETENŢE_PERSONAL care a produs adăugarea sau înregistrarea instanţei/liniei respective. Declanşatorul de inserare al tabelei-stăpân va aloca tranzacţiei, pe baza secvenţei, un identificator - `IdJurnalizare_Init` -, iar cel de ştergere va aloca, tot pe baza secvenţei, o valoare pentru atributul `IdJurnalizare_Fin`. Interesul nostru se leagă preponderent de modificările din tabela principală ; fiecare modificare generează în tabela-jurnal o ştergere, urmată de o inserare. Or, cheia problemei este ca ştergerea şi inserarea să aibă acelaşi identificator al tranzacţiei, adică, la ştergere, `IdJurnalizare_Fin` să fie identic cu `IdJurnalizare_Init` al noii instanţe inserate.

Listing 10.9. Un alt mod de jurnalizare

```
DROP TABLE ist_competente_personal ;

CREATE TABLE ist_competente_personal (
     idjurnalizare_init NUMBER(14) PRIMARY KEY,
     marca NUMBER (6) NOT NULL,
     idcompfrunza NUMBER (10) NOT NULL,
     nivel NUMBER(5) NOT NULL,
     data_init DATE DEFAULT CURRENT_DATE NOT NULL,
     data_fin DATE DEFAULT NULL,
     idjurnalizare_fin NUMBER(14) DEFAULT NULL
     ) ;

DROP SEQUENCE seq_idjurnalizare ;
CREATE SEQUENCE seq_idjurnalizare
    START WITH 1 MINVALUE 1 MAXVALUE 99999999999999
    NOCYCLE NOCACHE ORDER INCREMENT BY 1 ;

-------
CREATE OR REPLACE TRIGGER trg_competente_personal_ins
    AFTER INSERT ON competente_personal
    FOR EACH ROW
BEGIN
    INSERT INTO ist_competente_personal VALUES (seq_idjurnalizare.NextVal,
        :NEW.marca, :NEW.idcompfrunza, :NEW.Nivel, SYSDATE, NULL, NULL) ;
END ;
/

-----
CREATE OR REPLACE TRIGGER trg_competente_personal_del
    AFTER DELETE ON competente_personal
    FOR EACH ROW
BEGIN
    UPDATE ist_competente_personal
    SET data_fin = SYSDATE, idjurnalizare_fin = seq_idjurnalizare.NextVal
    WHERE marca=:OLD.marca AND idcompfrunza=:OLD.IdCompFrunza
        AND data_fin IS NULL ;
```

```
END ;
/

-----
CREATE OR REPLACE TRIGGER trg_competente_personal_upd
   AFTER UPDATE ON competente_personal
   FOR EACH ROW
DECLARE
   v_data DATE ;
BEGIN
   v_data := SYSDATE ;

   -- se declara "expirate" vechile valori
   UPDATE ist_competente_personal
   SET data_fin = v_data, idjurnalizare_fin = seq_idjurnalizare.NextVal
   WHERE marca=:OLD.marca AND idcompfrunza=:OLD.idCompFrunza
      AND data_fin IS NULL ;

   -- se insereaza noile valori
   INSERT INTO ist_competente_personal VALUES (seq_idjurnalizare.CurrVal,
      :NEW.marca, :NEW.idcompfrunza,
      :NEW.Nivel, v_data + 1 /(24*60*60), NULL, NULL) ;

END ;
/
```

Referindu-ne la problema noastră, dacă marca angajatului 20002 va fi modificată în 22222 la un moment dat: `UPDATE personal9 SET marca=22222 WHERE numepren='Mihuleac Iulian'`, operaţiunile se vor derula astfel:

- se denulizează valorile atributelor `Data_Fin` şi `IdJurnalizare_Fin` pentru toate liniile din IST_COMPETENŢE_PERSONAL în care valoarea mărcii este 20002; primul atribut ia valoarea datei/orei curente, iar al doilea „muşcă" valoarea următoare din secvenţă;
- se inserează un set de linii egal cu numărul celor modificate în COMPETENŢE_PERSONAL, în care valoarea `IdJurnalizare_Init` este identică cu cea a `IdJurnalizare_Fin` de la pasul anterior, iar `Data_Init` este valoarea `Data_Fin` de la pasul anterior plus o secundă.

Să presupunem că, după un timp, marca aceluiaşi angajat este modificată din nou, din 22222 în 22223: `UPDATE personal9 SET marca=22223 WHERE numepren = 'Mihuleac Iulian'`. În funcţie de momentul pentru care interesează nivelul competenţelor acestui angajat, valorile corecte vor fi furnizate fie de liniile în care marca este 20002, fie de cele în care marca este 222222, fie de cele în care marca este 22223. Ei bine, avem motive serioase de a privi languros oferta SQL de la Oracle, deoarece avem un excelent sprijin în astfel de situaţii. Indiferent de câte modificări s-au produs asupra mărcii, pornind de la marca actuală, putem şti toate valorile acestui atribut printr-o interogare ierarhică, în care legătura dintre niveluri se

realizează prin valoarile identice ale identificatorilor de „expirare" din vechile linii şi de adăugare a noilor instanţe:

```
SELECT *
FROM ist_competente_personal
START WITH marca IN (SELECT marca
       FROM personal9
       WHERE numepren='Mihuleac Iulian')
CONNECT BY PRIOR idjurnalizare_init=idjurnalizare_fin
```

Dacă dorim să aflăm nivelul tuturor competenţelor actuale (presupunem că nu s-a modificat nimic în nomenclatorul competenţelor) pentru angajatul Mihuleac Iulian la 16 februarie 2005, ora 10:25, interogarea ce foloseşte artificiul de mai sus ar fi:

```
SELECT dencompetenta, nivel
FROM competente c INNER JOIN
   (SELECT *
    FROM
      (SELECT *
      FROM ist_competente_personal
      START WITH marca IN (SELECT marca
         FROM personal9
         WHERE numepren='Mihuleac Iulian')
      CONNECT BY PRIOR
         idjurnalizare_init=idjurnalizare_fin
      )
   WHERE  TO_DATE('2005-02-16 10:25','YYYY-MM-DD HH24:MI')
      BETWEEN data_init AND NVL(data_fin, SYSDATE)
   ) cp ON c.idcompetenta=cp.idcompfrunza
```

Varianta sută la sută corectă a acestei interogări ar trebui să lucreze cu tabela IST_COMPETENŢE, o subconsultare similară celei aplicate IST_COMPETENŢE_PERSONAL extrăgând elementele definitorii ale competenţelor la momentul *16 februarie 2005, ora 10:25*. În acest stil putem rezolva orice problemă legată de temporalitate, chiar dacă preţul plătit nu este chiar neglijabil.

10.3. Protejarea „trecutului". Închideri contabile şi nu numai

Mare parte dintre proiectanţii şi dezvoltatorii de aplicaţii economice nu iau prea în serios problemele ce ţin de legislaţia afacerilor, care interzice modificarea ulterioară a unora dintre datele privitoare la operaţiuni consemnate în documente oficiale. Mai precis, o dată ce o firmă a depus la administraţiile financiare balanţa de verificare

sau bilanţul, îi este interzis să mai modifice retroactiv datele respective, chiar dacă descoperă greşeli. Corectarea se realizează, de obicei, sub forma stornării urmate de preluarea sumelor corecte, ambele operaţiuni fiind consemnate în contul lunii sau lunilor următoare descoperirii erorii.

În ceea ce ne priveşte, vom examina în acest paragraf două tipuri de „interdicţii temporale". Prima priveşte eternele şi fascinantele facturi emise. O factură emisă consemnează o vânzare de produse şi servicii. Fiind un document fundamental în relaţiile dintre partenerii de afaceri, având regim de probă în justiţie, ca să nu mai vorbim de apetitul fiscului în materie de impozite şi taxe, factura este unul dintre cele mai sensibile documente întocmite de firmă, având un regim de întocmire şi circulaţie foarte strict. De aceea, după redactarea facturii şi listarea sa se face o verificare temeinică a datelor sale, după care factura îşi începe circuitul. Acesta este momentul după care factura nu trebuie să mai sufere nici o modificare!

Din perspectiva bazei de date, introducerea unei facturi înseamnă inserarea unei linii în tabela FACT, apoi a *n* tupluri în LINII_FACT, corespunzătoare celor *n* produse şi servicii vândute. După inserarea celor *n* linii în LINII_FACT, ar mai putea surveni modificări în ambele tabele, până la confirmarea facturii, care semnalează definitivarea sa. Problema e că, în actuala structură, momentul definitivării facturii nu este preluat în tabelele bazei de date VÂNZĂRI. O soluţie destul de simplă ar fi să recurgem la un atribut special – `Definitivată` – pe care să-l introducem în tabela FACT:

```
ALTER TABLE fact ADD definitivata CHAR(1) DEFAULT 'N'
   NOT NULL CHECK (definitivata IN ('D','N')) ;
```

Prin interfaţa aplicaţiei, utilizatorul va semnala definitivarea şi „închiderea" facturii, valoarea atributului `Definitivată` devenind *D* (da). Interzicerea oricărei modificări în factură sau a ştergerii facturii va cădea în seama declanşatoarelor tabelelor FACT şi LINII_FACT, care vor verifica starea noului atribut. *Listing*-ul 10.10 conţine banalul declanşator de ştergere al tabelei FACT.

Listing 10.10. Blocarea ştergerii unei facturi definitivate

```
CREATE OR REPLACE TRIGGER trg_fact_del
   BEFORE DELETE ON fact FOR EACH ROW
BEGIN
   IF :OLD.definitivata='D' THEN
      RAISE_APPLICATION_ERROR(-20349, 'Nu puteti sterge o factura definitivata!');
   END IF ;
END ;
/
```

Lucrurile nu sunt prea diferite pentru declanşatorul de modificare, iar declanşatoarele de inserare, modificare şi ştergere în LINII_FACT ar putea beneficia de o funcţie specială care să returneze valoarea atributului-stare. Atenţie! Dacă tabela FACT conţine şi atributul calculat `ValÎncasată`, acesta are un regim special, putând fi editat până la încasarea totală a facturii.

Cei care cunosc, fie şi sumar, practica financiar-contabilă ar putea obiecta: bine-bine, dar dacă din momentul confirmării facturii nu mai putem face nici o modificare, cum consemnăm faptul că factura respectivă a fost, peste câteva ore, zile, săptămâni, chiar luni, stornată? Mai ales că ar trebui să consemnăm atât momentul stornării, cât şi, eventual, noua factură ce reglează stornarea. Variante de lucru:

- în FACT introducem atributele `DataStornării` şi `NrFactCorectoare`, iar prin declanşatoare autorizăm modificarea postdefinitivarea numai a acestor două câmpuri;
- în FACT introducem atributul `EsteStornată` cu valori D/N şi, în plus, creăm o tabelă specială FACT_STORNATE (`NrFact`, `DataStornării`, `NrFactCorectoare`);
- păstrăm neschimbată structura tabelei FACT, gestionând stornarea exclusiv prin tabela FACT_STORNATE (`NrFact`, `DataStornării`, `NrFactCorectoare`).

Ultima variantă este cea mai puţin traumatizantă pentru declanşatoarele bazei, însă are marele dezavantaj că, la obţinerea valorile exacte ale vânzărilor, trebuie făcută, de fiecare dată, joncţiunea externă a tabelei FACT cu FACT_STORNATE.

Al doilea tip de „interdicţie temporală" este tocmai cel invocat în primele fraze ale paragrafului – închiderea contabilă. La finalul unei luni calendaristice sau în primele zile ale lunii următoare are loc operaţiunea creatoare de frisoane pentru orice contabil care se respectă – închiderea de lună. Cei care au avut ocazia să lucreze într-o firmă/organizaţie ştiu că în anumite zile din lună în biroul *Contabilitate* se merge pe vârfuri, iar câteva persoane cu feţe congestionate nu pot fi deranjate nici în ruptul capului, deoarece „fac închiderea"! Dincolo de legende şi datini populare contabile, închiderea presupune centralizarea operaţiunilor pe module ale sistemului informaţional financiar-contabil (imobilizări, stocuri, încasări/plăţi, salarizare etc.), calculul dărilor către stat (TVA, impozite şi contribuţii legate de salarizare), descărcarea conturilor de cheltuieli şi venituri etc.

Din momentul terminării închiderii, listării/redactării şi depunerii la fisc a tuturor documentelor, se interzice modificarea retroactivă a datelor conţinute în rapoartele respective. Lucrurile sunt absolut normale, deoarece orice astfel de modificare ar putea antrena decalaje informaţionale între datele din bază şi documentele oficiale. De aceea, este nevoie de o tabelă specială pe care o vom numi ÎNCHIDERI.

```
CREATE TABLE inchideri (
   an NUMBER(4) DEFAULT EXTRACT (YEAR FROM CURRENT_DATE)
      NOT NULL CHECK (an >= 2004),
   luna NUMBER(2) DEFAULT
      EXTRACT (MONTH FROM CURRENT_DATE)
      NOT NULL CHECK (luna BETWEEN 1 AND 12),
   datainchiderii DATE DEFAULT CURRENT_DATE NOT NULL,
   operator VARCHAR2(30) DEFAULT USER,
   PRIMARY KEY (an, luna)
   ) ;
```

La momentul finalizării închiderii pe luna februarie 2005, se va insera o linie în această tabelă:

```
INSERT INTO inchideri (an, luna) VALUES (2005,2);
```

Orice inserare, modificare sau ştergere care antrenează valori trebuie supusă controlului „temporal"; astfel, la modificarea valorii sau TVA-ului unei facturi, trebuie verificat dacă respectiva factură nu face parte dintr-o lună pentru care s-a făcut închiderea, caz în care inserarea/modificarea/ştergerea respectivă trebuie anulată. Pentru mărirea vitezei de lucru pachetul `pac_închideri` din *listing*-ul 10.11 foloseşte o variabilă publică iniţializată la începutul sesiunii şi la o inserare în tabela ÎNCHIDERI. Banalul declanşator de ştergere în tabela FACT va verifica, suplimentar, dacă data facturii întocmite se situează pe o lună deja „închisă", caz în care va declanşa o eroare.

Listing 10.11. Pachetul şi un declanşator dedicate închiderilor contabile

```
-------------------------------------------------------------
-- pachetul PAC_INCHIDERI
CREATE OR REPLACE PACKAGE pac_inchideri AS
    v_ultima_zi_inchisa DATE := NULL ;
    PROCEDURE init_v_ultima_zi_inchisa ;
END ;
/
CREATE OR REPLACE PACKAGE BODY pac_inchideri AS
--------------------------------
PROCEDURE init_v_ultima_zi_inchisa
IS
    v_data DATE ;
BEGIN
    IF v_ultima_zi_inchisa  IS NULL THEN
        SELECT MAX(TO_DATE('01/'|| luna || '/' || an, 'DD/MM/YYYY'))
        INTO v_data FROM inchideri ;
        v_ultima_zi_inchisa := LAST_DAY(v_data) ;
    END IF ;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        v_ultima_zi_inchisa := DATE'1999-12-31' ;
END ;
END ; -- pachet
/
-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_inchideri_ins
    AFTER INSERT ON inchideri
BEGIN
    pac_inchideri.v_ultima_zi_inchisa := NULL ;
    pac_inchideri.init_v_ultima_zi_inchisa ;

END ;
/
```

```
-------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_del
   BEFORE DELETE ON fact FOR EACH ROW
BEGIN
   IF :OLD.definitivata='D' THEN
      RAISE_APPLICATION_ERROR(-20349, 'Nu puteti sterge o factura definitivata!');
   END IF ;

   pac_inchideri.init_v_ultima_zi_inchisa ;

   IF pac_inchideri.v_ultima_zi_inchisa >= :OLD.DataFact THEN
      RAISE_APPLICATION_ERROR(-20899,
         'Factura apartine unei luni <<inchise>>!') ;
   END IF ;
END ;
/
```

10.4. Arhivări, fuziuni, compatibilităţi

Chiar şi în SGBD-urile generoase spaţiul este limitat, iar viteza de interogare, vitală pentru bunul mers al aplicaţiei. Interesul unei scheme de stocare este de a prelua toate informaţiile relevante în vederea consultării lor ulterioare. Cu cât datele preluate sunt mai analitice şi numărul trazacţiilor companiei ce deţine baza de date este mai ridicat, cu atât dimensiunea bazei devine tot mai mare, atingând, nu de puţine ori, în câţiva ani zeci şi sute de gigaocteţi, poate chiar mai mult. Dacă în paragraful 9.7 ne puneam problema ruperii relaţiilor „pe verticală" în vederea creşterii vitezei de acces prin gruparea atributelor în tabele diferite (în funcţie de frecvenţa folosirii atributelor), acum ne gândim să rupem pe orizontală tabelele „tranzacţionale", prin mutarea liniilor referitoare la operaţiuni finalizate cu ceva timp în urmă.

Să luăm cazul bazei de date VÂNZĂRI. Aplicaţia a fost implementată în 1997 şi, dacă nu recurgem la arhivare, aflarea valorilor facturate şi încasate ale unui client în mai 2005 prespune consultatea unor tabele FACT, LINII_FACT, ÎNCAS_FACT uriaşe. Chiar dacă SGBD-urile dispun de mecanisme optimizatoare ale accesului, este de aşteptat ca o tabelă de ordinul gigaocteţilor să presupună o viteză de consultare sensibil mai mică decât una de dimensiunea câtorva megaocteţi.

Practic, datele din perioada 1997-2003 sunt mult mai rar consultate decât cele din ultimele 12 luni. Tocmai luând în calcul o viteză de acces cât mai bună, putem să apelăm la o serie de tabele pentru arhivare, cum ar fi FACT_ARHIVATE, LINII_FACT_ARHIVATE. Operaţiunea de arhivare ar presupune vărsarea tuturor liniilor până la o dată specificată – vezi *listing*-ul 10.12.

Listing 10.12. Arhivarea facturilor

```
DROP TABLE linii_fact_arhivate ;
DROP TABLE incas_fact_arhivate ;
DROP TABLE incasari_arhivate ;
DROP TABLE fact_arhivate ;
CREATE TABLE fact_arhivate AS SELECT * FROM fact WHERE 1=2 ;
CREATE TABLE linii_fact_arhivate AS SELECT * FROM linii_fact WHERE 1=2 ;
CREATE TABLE incasari_arhivate AS SELECT * FROM incasari WHERE 1=2 ;
CREATE TABLE incas_fact_arhivate AS SELECT * FROM incas_fact WHERE 1=2 ;

ALTER TABLE fact_arhivate ADD DataArhivarii DATE ;
ALTER TABLE fact_arhivate ADD PRIMARY KEY (NrFact) ;

ALTER TABLE linii_fact_arhivate ADD DataArhivarii DATE ;
ALTER TABLE linii_fact_arhivate ADD PRIMARY KEY (NrFact, Linie) ;

ALTER TABLE incasari_arhivate ADD DataArhivarii DATE ;
ALTER TABLE incasari_arhivate ADD PRIMARY KEY (CodInc) ;

ALTER TABLE incas_fact_arhivate ADD DataArhivarii DATE ;
ALTER TABLE incas_fact_arhivate ADD PRIMARY KEY (CodInc, NrFact) ;


CREATE OR REPLACE PROCEDURE p_arhivare_fact (datafact_ DATE)
IS
BEGIN
   -- se adauga liniile de arhivat in tabele-arhiva
   INSERT INTO linii_fact_arhivate
      SELECT linii_fact.*, SYSDATE
      FROM linii_fact
      WHERE nrfact IN
         (SELECT nrfact FROM fact
            WHERE datafact <= datafact_) ;
   INSERT INTO fact_arhivate
      SELECT fact.*, SYSDATE FROM fact WHERE datafact <= datafact_ ;

   INSERT INTO incasari_arhivate
      SELECT incasari.*, SYSDATE FROM incasari
      WHERE codinc IN
         (SELECT codinc FROM incas_fact WHERE nrfact IN
         (SELECT nrfact FROM fact WHERE datafact <= datafact_)
            );
   INSERT INTO incas_fact_arhivate
      SELECT incas_fact.*, SYSDATE FROM incas_fact
      WHERE nrfact IN (SELECT nrfact FROM fact
            WHERE datafact <= datafact_) ;

   -- se dezactiveaza declansatoarele, pentru a evita verificarile suplimentare
   --  si actualizarea atributelor calculate
   EXECUTE IMMEDIATE ' ALTER TABLE linii_fact DISABLE ALL TRIGGERS ' ;
```

```
    EXECUTE IMMEDIATE ' ALTER TABLE incas_fact DISABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE ' ALTER TABLE fact DISABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE ' ALTER TABLE incasari DISABLE ALL TRIGGERS ' ;

    -- se sterg liniile arhivate
    DELETE FROM linii_fact   WHERE nrfact IN
       (SELECT nrfact FROM fact WHERE datafact <= datafact_) ;
    DELETE FROM incas_fact WHERE nrfact IN
       (SELECT nrfact FROM fact WHERE datafact <= datafact_) ;
    DELETE FROM incasari WHERE codinc IN
       (SELECT codinc FROM incasari_arhivate) ;

    DELETE FROM fact WHERE datafact <= datafact_ ;

    -- se reactiveaza declansatoarele
    EXECUTE IMMEDIATE 'ALTER TABLE linii_fact ENABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE 'ALTER TABLE fact ENABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE 'ALTER TABLE incasari ENABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE 'ALTER TABLE incas_fact ENABLE ALL TRIGGERS ' ;
END ;
```

În condiţiile în care tabelele-arhivă vor fi protejate la ştergeri şi modificări, rezultă un spor de viteză considerabil, deoarece tabelele sunt mult mai „suple", păstrând numai informaţii recente. Atenţie ! Arhivarea nu trebuie operată prea des, iar facturile arhivate trebuie să fi fost reglate (încasate) definitiv !

Marea problemă a acestui stil de lucru este că rapoartele şi interogările privind vânzările şi încasările trebuie să-şi extragă informaţiile atât din tabelele FACT, LINII_FACT..., cât şi din arhivele lor. Se pot crea, în acest scop, tabele virtuale, după modelul :

```
CREATE VIEW vFact AS
   SELECT fact.*, SYSDATE AS DataCurenta
   FROM fact
      UNION
   SELECT *
   FROM fact_arhivate
```

În continuare, să presupunem că două firme informatizate până în dinţi decid să fuzioneze. În urma fuziunii, şi bazele de date se vor uni, ce-i drept nici dintr-o sorbire, nici dintr-un click de mouse. Dacă schemele sunt similare, şi sunt motive să credem că, pentru unele aplicaţii precum VÂNZĂRI, supoziţia nu este chiar neacoperită în practică, ce se întâmplă cu entităţile comune ? Cu profiluri similare, cele două firme au, mai mult ca sigur, clienţi comuni şi produse comune. Este evident că după fuziune trebuie să rămână o singură instanţă a fiecărei entităţi, altminteri identificarea ar fi imposibilă, iar informaţiile extrase eronate.

Spre exemplu, cele două firme aveau fiecare câte o tabelă de clienţi pe care le vom nota CLIENŢI1 şi CLIENŢI2. Codurile clienţilor nu se suprapun, însă firma

partener ce este identificată prin codul 2011 în CLIENŢI1 este aceeaşi cu cea desemnată de codul 1001 în CLIENŢI2. Trebuie, aşadar, să renunţăm la una dintre instanţe. Scenariul n-ar fi prea complicat:

```
-- se "varsă" CLIENTI2 în CLIENTI1 şi procedăm analog cu
--   tabelele copil, nepot etc.: FACTURI, LINIFACT…
INSERT INTO clienti1 SELECT * FROM clienti2 ;
INSERT INTO facturi1 SELECT * FROM facturi2 ;
-- …
--

-- se modifică în tabelele copil valorile 2011 ale CodCl în 1001
UPDATE facturi1 SET codcl=1011 WHERE codcl=2011
-- … se continuă cu ceilalţi copii ai tabelei CLIENTI

-- se şterge din CLIENTI1 linia clientului 2011
DELETE FROM clienti1 WHERE codcl=2011
```

Bun, şi acum vestea proastă: cele două firme sunt, de fapt, lanţuri de supermagazine care şi-au implementat sistemele informaţionale de ani buni, iar datele adunate sunt atât de voluminoase, încât nu încap în întregime pe hard-disk, găsindu-se doar pe câteva sute de CD-ROM-uri. Şi, chiar dacă ar încăpea pe un întreg hard-disk, nu avem nici resurse şi nici timp pentru a le copia, actualiza şi rescrie.

Am luat acest exemplu pentru a vă intimida, însă avem şi un altul mult mai banal. Deseori, atunci când o aplicaţie este exploatată de mai mulţi utilizatori, se întâmplă ca, din neatenţie, unul dintre ei să nu sesizeze că clientul din factură există deja (poate nici nu se vede prea bine numele acestuia pe factură) în bază, alege butonul *Client nou* al formularului şi... gata, în câteva zeci de secunde ne alegem cu un client aflat pe două linii ale tabelei CLIENŢI. Iar dacă samavolnicia e identificată mult mai târziu, după salvarea pe CD a bazei de date şi ştergerea înregistrărilor salvate, avem o situaţie ce nu diferă prea mult de cea descrisă anterior.

Fireşte cea mai bună pare a fi soluţia de jurnalizare a modificărilor din paragraful 10.2, ce ar putea fi reexaminată şi rescrisă pentru a rezolva şi această problemă. Putem apela însă şi la o altă variantă, mai simplă, în care să stocăm corespondenţele codurilor clienţilor: CODURI_CLIENTI_SCHIMBATE {`CodClActual`, `UnCodClAnterior`, `DataConsemnarii`}.

Grija cea mai mare a soluţiei este ca nici o valoare a `CodCl` să nu se recicleze. O dată folosită, din momentul renunţării la ea, valoarea trebuie să rămână blocată, altminteri confuziile ar fi iminente. În ultima propoziţie din paragraful 9.5 ameninţam că vom reveni cu câteva detalii legate de folosirea de relaţii redundante pentru gestionarea corespondenţelor dintre datele actuale şi cele anterioare. Or, un asemenea tip de tabelă ne-ar putea rezolva problema evitării reciclărilor ulterioare, mai ales atunci când o parte dintre arhive nici nu mai sunt în baza de date curentă, ci arhivate pe CD-uri:

CLIENŢI_ARHIVAŢI (CodCl, DataArhivării)
FACT_ARHIVATE (NrFact, CodCl, DataArhivării)

Inserările şi modificările CodCl şi NrFact în CLIENŢI şi FACT vor trebui monitorizate, declanşatoarele celor două tabele consultând aceste două tabele arhivă, evitându-se, astfel, valorile ce au fost deja salvate şi arhivate.

Bibliografie

[Abiteboul & Bidoit 84] Abiteboul, S., Bidoit, N., *Non First Normal Form Relations to Represent Hiearchically Organized Data*, Proc. of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, Waterlov, Ontario, Canada, 1984

[Aho ş.a. 79] Aho, A.V., Beeri, C., Ullman, J.D., „The theory of joins in relational databases", *ACM Transactions on Database Systems*, vol. 4, nr. 3, septembrie 1979

[Atzeni ş.a. 99] Atzeni, P., Ceri, S., Paraboschi, S., Torlone, R., *Database Systems. Concepts, Languages, and Architectures*, McGraw-Hill, London, 1999

[Adiba & Delobel 82] Adiba, M., Delobel, C., *Bases de données et systèmes relationnels*, Éditions Dunod, 1982

[Atkinson ş.a. 89] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonick, D., *The Object-Oriented Database System Manifesto*, International Conference on Deductive and Object-Oriented Databases, Kyoto, 1989

[Bâscă 97] Bâscă, O., *Baze de date*, Editura All, Bucureşti, 1997

[Bernstein 76] Bernstein, P.A., „Synthesizing Third Normal Form Relations from Functional Dependencies", *ACM Transactions on Database Systems*, vol. 1, nr. 4, decembrie 1976

[Bock & Schrage 02] Bock, D.B., Schrage, J.F., „Denormalization Guidelines for Base and Transaction Tables", *Inroads - SIGCSE Bulletin*, vol. 34, nr. 2, 2002

[Casanova ş.a. 82] Casanova, M.A., Fagin, R., Papadimitrou, C.H., *Inclusion dependencies and their interaction with functional dependencies*, Proc. of the 1st ACM SIGACT--SIGMOD symposium on Principles of database systems, Los Angeles, 1982

[Celko 99] Celko, J., *Data & Databases: Concepts in Practice*, Morgan-Kaufmann, San Francisco, 1999

[CODASYL 69] CODASYL Systems Committee, *A Survey of Generalized Data Base Management Systems*, Technical Report, mai 1969

[Codd 69] Codd, E.F., *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ 599, august 1969

[Codd 70] Codd, E.F., „A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, vol. 13, nr. 6, 1970

[Codd 72] Codd, E.F., „Further Normalization of the Database Relational Model", *DataBase Systems*, Courant Computer Science Symposia Series, vol. 6, Englewood Cliffs, N.J., Prentice-Hall, 1972

[Codd 74] Codd, E.F., *Recent Investigations into Relational Database Systems*, Proc. IFIP Congress, Stockholm, 1974

[Codd 79] Codd, E.F., „Extending the Database Relational Model to Capture More Meaning", *ACM Transactions on Database Systems*, vol. 4, nr. 4, decembrie 1979

[Codd 90] Codd, E.F., *The Relational Model for Database Management. Version 2*, Addison--Wesley, 1990

[Connoly ş.a. 96] Connolly, T., Begg, C.E., Strachan, A.D., *Database Systems. A practical Approach to Design, Implementation and Management*, Addison-Wesley, Harlow, 1996

[Connoly & Begg 00] Connolly, T., Begg, C.E., *Database Solutions. A step-by-step guide to building databases*, Addison-Wesley, Harlow, 2000

[Connoly & Begg 02] Connolly, T., Begg, C.E., *Database Systems. A practical Approach to Design, Implementation and Management*, ediţia a III-a, Addison-Wesley, Harlow, 2002

[Date 86] Date, C.J., *An Introduction to Database Systems*, Addison-Wesley, Reading, Massachusetts, ediţia a IV-a, 1986

[Date 99] Date, C.J., „Thirty Years of Relational : The First Three Normal Forms", *Intelligent Enterprise*, vol. 2, nr. 5-6 (martie-aprilie) 1999

[Date 00] Date, C.J., *An Introduction to Database Systems*, ediţia a VII-a, Addison-Wesley, Reading, Massachusetts, 2000

[Date 03] Date, C.J., „What First Normal Form Really Means", *http : //www.dbdebunk.com*, 2003 (ultima accesare - martie 2004)

[Date 04] Date, C.J., *An Introduction to Database Systems*, ediţia a VIII-a, Pearson Addison-Wesley, Boston, 2004

[Date & Darwen 00] Date, C.J., Darwen, H., *Foundation for Database Systems. The Third Manifesto*, ediţia a II-a, Addison-Wesley, Reading, Massachusetts, 2000

[Delobel ş.a. 91] Delobel, C., Leclus, C., Richard, P., *Bases de données. De systèmes relationnels aux systèmes à objets*, InterÉditions, Paris, 1991

[Dollinger 98] Dollinger, R., *Baze de date şi gestiunea tranzacţiilor*, Editura Albastră, Cluj-Napoca, 1998

[Dreizen & Chang 89] Dreizen, H.M., Chang, S.K., „Imprecise Schema : A Rationale for Relations with Embedded Subrelations", *ACM Transactions on Database Systems*, vol. 14, nr. 4, decembrie 1989

[Elmasri & Navathe 00] Elmasri, R., Navathe, S.R., *Fundamentals of Database Systems*, Addison-Wesley, Reading, Massachusetts, 2000

[Fagin 77] Fagin, R., „Multivalued Dependencies and a New Normal Form for Relational Databases", *ACM Transactions on Database Systems*, vol. 2, nr. 3, septembrie 1977

[Fagin 79] Fagin, R., *Normal Forms and Relational Database Operators*, Proc. of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, 1979

[Fagin 81] Fagin, R., „A Normal Form for Relational Databases That Is Based on Domains and Keys", *ACM Transactions on Database Systems*, vol. 6, nr. 3, septembrie 1981

[Fagin ş.a. 82] Fagin, R., Mendelzon, A., Ullman, J., „A Simplified Universal Relation Assumption and Its Properties", *ACM Transactions on Database Systems*, vol. 7, nr. 3, septembrie 1982

[Fleming & von Halle 89] Fleming, C., von Halle, B., *Handbook of Relational Database Design*, Addison-Wesley, Reading, Massschusetts, 1989

[Florescu ş.a. 99] Florescu, V., Stanciu, V., Cozgarea, G., Cozgarea, A., *Baze de date*, Editura Economică, Bucureşti, 1999

[Fotache 97] Fotache, M., *Baze de date relaţionale. Organizare, interogare şi normalizare*, Editura Junimea, Iaşi, 1997

[Fotache 00] Fotache, M., „Despre prima formă normală", *PC Report*, nr. 92, mai 2000

[Fotache 01-1] Fotache, M., „Proiectarea bazelor de date. Câteva probleme ale formelor normalizate 2 şi 3", *NET Report*, nr. 104, mai 2001

[Fotache 01-2] Fotache, M., „Proiectarea bazelor de date. Cui îi mai este frică de formele normalizate Boyce-Codd, 4 şi 5 ?", *NET Report*, nr. 105, iunie 2001

[Fotache ş.a. 01-3] Fotache, M., *SQL. Dialecte DB2, Oracle şi Visual FoxPro*, Editura Polirom, Iaşi, 2001

[Fotache ş.a. 02-1] Fotache, M., Brava, I., Strîmbei, C., Creţu, L., *Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale*, Editura Polirom, Iaşi, 2002

[Fotache ş.a. 02-2] Fotache, M., Strîmbei, C., Creţu, L., „Baze de date - între orientări teoretice şi piaţa dezvoltării de aplicaţii", în [Oprea ş.a. 02], pp. 224-276

[Fotache ş.a. 03] Fotache, M., Strîmbei, C., Creţu, L., *Oracle 9i2. Ghidul dezvoltării aplicaţiilor profesionale*, Editura Polirom, Iaşi, 2003

[Fotache 04-1] Fotache, M., „Pe marginea şi înăuntrul primei forme normale", *Informatică economică*, vol. 8, nr. 3, 2004

[Fotache 04-2] Fotache, M., „Do practitioners need database normalization?", *Proc. of the Central and East European Conference in Business Information Systems*, Editura Risoprint, Cluj-Napoca, 2004

[Fotache 04-3] Fotache, M., „Trei dintre neclarităţile normalizării", *Informatică economică*, vol. 8, nr. 4, 2004

[Fotache 05] Fotache, M., *Database Designers and Normalization : Anathomy of a Divorce*, Proc. of BIS2005 - 8th International Conference on Business Information Systems, Poznan, 2005

[Garcia-Molina ş.a. 02] Garcia-Molina, H., Ullman, J., Widom, J., *Database Systems. The complete Book*, Prentice Hall, Upper Saddle Riner, New Jersey, 2002

[Gardarin 91] Gardarin, G., *Bases de données. Les systèmes et leurs langages*, Eyrolles, Paris, 1991

[Gillette ş.a. 95] Gillette, R., Muench, D., Tabaka, J., *Pshysical Database Design for Sysbase SQL Server*, Prentice-Hall, Englewood Cliffs, N.J., 1995

[Harrington 02] Harringtom, J.L., *Relational Database Design Clearly Explained*, ediţia a II-a, Morgan Kaufman, Amsterdam, 2002

[Hoberman 02] Hoberman, S., „The Denormalization Survival Guide", *The Data Administration Newsletter*, iulie 2002

[Hoffer ş.a. 02] Hoffer, J.A., Prescott, M.B., McFadden, F.R., *Modern Database Management*, Prentice-Hall (Pearson Education), 2002

[Johnson & Klug 82] Johnson, D.S., Klug, A., *Testing containment of conjuctive queries under functional and inclusion dependencies*, Proc. of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems, Los Angeles, 1982

[Kent 81] Kent, W., „Consequences of Assuming a Universal Relation", *ACM Transactions on Database Systems*, vol. 6, nr. 4, decembrie 1981

[Kent 83-1] Kent, W., „A Simple Guide to Five Normal Forms in Relational Database Theory", *Communications of the ACM*, vol. 26, nr. 2, februarie 1983

[Kent 83-2] Kent, W., „The Universal Relation Revisited", *ACM Transactions on Database Systems*, vol. 8, nr. 4, decembrie 1983

[Korth & Silberschatz 88] Korth, H.F., Silberschatz, A., *Systèmes de gestion de bases de données*, McGraw-Hill, Paris, 1988

[Lonigro 98] Lonigro, M., *The Case for the Surrogate Key*, DataBase Programming & Design, mai 1998 (*online* : *http : //www.dbpd.com/vault/9805xtra.htm*)

[Lungu ş.a. 95] Lungu, I., Bodea, C, Bădescu, G., Ioniţă, C., *Baze de date. Organizare, proiectare şi implementare*, Editura All, Bucureşti, 1995

[Miranda & Busta 90] Miranda, S., Busta, J.M., *L'art des bases de données*, vol. I-II, Eyrolles, Paris, 1990

[Muller 99] Muller, R., *Database Design for Smartles. Using UML for Data Modeling*, Morgan Kaufmann, San Francisco, California, 1999

[Mullins 98] Mullins, C., „Denormalization Guidelines", *The Data Administration Newsletter*, mai 1998

[Nicolas 78] Nicolas, J.M., *Mutual Dependencies and Some Results on Undecomposable Relations*, Proc. of the 4th International Conference on Very Large Data Bases, 1978

[O'Neil & O'Neil 01] O'Neil, P., O'Neil, E., *Database. Principles. Programming, Performance*, Morgan Kaufmann, San Francisco, 2001

[Oprea 99] Oprea, D., *Analiza sistemelor informaţionale*, Editura Polirom, Iaşi, 1999

[Oprea ş.a. 02] Oprea, D., Airinei, D., Fotache, M. (coordonatori), *Sisteme informaţionale pentru afaceri*, Editura Polirom, Iaşi, 2002

[Pascal 00] Pascal, F., *Practical Issues in Database Management: A Reference for the Thinking Practitioner*, Addison-Wesley, 2000

[Pascal 01] Pascal, F., „Denormalization for Performance - Et Tu Academia?", *http://www.dbdebunk.com*, iulie 2001

[Pascal 03] Pascal, F., „What First Normal Form Really Means Not", *http://www.dbdebunk.com*, 2003 (ultima accesare - martie 2004)

[Poolet 00] Poolet, M., „Responsable Denormalization. How to break the rules and get away with it", *SQLServer Magazine*, octombrie 2000

[Popescu 96] Popescu, I., *Baze de date relaţionale*, Editura Universităţii Bucureşti, 1996

[Popescu 01] Popescu, I., *Modelarea bazelor de date*, Editura Tehnică, Bucureşti, 2001

[Pratt & Adamski 91] Pratt, P.J., Adamski, J.J., *Database Systems. Management and Design*, ediţia a II-a, Boyd & Fraser, Boston, 1991

[Riordan 99] Riordan, R., *Designing Relational Database Systems*, Microsoft Press, 1999

[Rissanen 77] Rissanen, J., „Independent Components of Relations", *ACM Transactions on Database Systems*, vol. 2, nr. 4, decembrie 1977

[Root 01] Root, D., „Denormalization and the Rules of Reconstruction", *The Data Administration Newsletter*, ianuarie 2001

[Saleh 94] Saleh, I., *Les bases de données relationnelles. Conception et réalisation*, Hermes, Paris, 1994

[Sciore 83] Sciore, E., *Inclusion Dependencies and the Universal Instance*, Proc. of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems, Atlanta, Georgia, 1983

[Simsion 01] Simsion, G., *Data Modeling Essentials. Analysis, Design, and Innovation*, ediţia a II-a, Coriolis, Scottsdale, Arizona, 2001

[Smith 85] Smith, H.C., „Database Design: Composing Fully Normalized Tables From a Rigourous Dependency Diagram", *Communication of the ACM*, vol. 28, nr. 8, august 1985

[Smith & Smith 77-1] Smith, J.M., Smith, D., „Database Abstractions: Aggregation", *Communication of the ACM*, vol. 20, nr. 6, iunie 1977

[Smith & Smith 77-2] Smith, J.M., Smith, D., „Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, vol. 2, nr. 6, iunie 1977

[Teorey 99] Teorey, T.J., *Database Modeling & Design*, Morgan Kaufmann, San Francisco, 1999

[Ullman 82] Ullman, J., *The U.R. Strikes Back*, Proc. of the 1st ACM SIGACT-SIGMID symposium on Principles of database systems, Los Angeles, 1982

[Ullman 83] Ullman, J., „On Kent's Consequences of Assuming a Universal Relation", *ACM Transactions on Database Systems*, vol. 8, nr. 4, decembrie 1983

[Wu 92] Wu, M.S., *The Practical Need for Fourth Normal Form*, Proc. of the 23rd ACM SIGCSE technical symposium on Computer Science education, Kansas City, Missouri, 1992

[Zaniolo & Melkanoff 81] Zaniolo, C., Melkanoff, M., „On the Design of Relational Database Schemata", *ACM Transactions on Database Systems*, vol. 6, nr. 1, martie 1981

La Editura POLIROM
au apărut:

Doina Hrinciuc Logofătu – *C++. Probleme rezolvate şi algoritmi*

Marin Fotache – *SQL. Dialecte DB2, Oracle şi Visual FoxPro*

Sabin Buraga, Gabriel Ciobanu – *Atelier de programare în reţele de calculatoare*

Marin Fotache, Ioan Brava, Cătălin Strîmbei, Liviu Creţu – *Visual FoxPro. Ghidul dezvoltării aplicaţiilor profesionale*

Sabin Buraga, Victor Tarhon-Onu, Ştefan Tanasă – *Programare Web în bash şi Perl*

Dragoş Acostăchioaie – *Administrarea şi configurarea sistemelor Linux*

Dragoş Acostăchioaie – *C şi C++ pentru Linux*

Sabin Buraga – *Proiectarea siturilor Web. Design şi funcţionalitate*

Călin Ioan Acu – *Optimizarea paginilor Web*

Ştefan Tanasă, Cristian Olaru, Ştefan Andrei – *JAVA de la 0 la expert*

Marin Fotache, Cătălin Strîmbei, Liviu Creţu – *Oracle 9i2. Ghidul dezvoltării aplicaţiilor profesionale*

Mihai Cioată – *ActiveX. Concepte şi aplicaţii*

Ştefan Trăuşan-Matu – *Programare în Lisp. Inteligenţă artificială şi web semantic*

Adrian Munteanu, Valerică Greavu-Şerban, Gabriel Cristescu – *Reţele Windows. Servere şi clienţi. Exemple practice*

Octavian Dospinescu – *Dezvoltarea aplicaţiilor în VisualBasic.NET*

Cristian A. Giumale – *Introducere în analiza algoritmilor. Teorie şi aplicaţie*

Dragoş Acostăchioaie – *FreeBSD*

Valeriu Iorga, Cristian Opincaru, Corina Stratan, Alexandru Chiriţă – *Structuri de date şi algoritmi. Aplicaţii în C++ folosind STL*

Marin Fotache – *Proiectarea bazelor de date. Normalizare şi postnormalizare. Implementări SQL şi Oracle*

www.polirom.ro

Coperta: Ionuţ Broştianu
Tehnoredactor: Lucian Pavel

Bun de tipar: martie 2005. Apărut: 2005
Editura Polirom, B-dul Carol I, nr. 4 • P.O. Box 266, 700506, Iaşi
Tel. & Fax: (0232) 21.41.00; (0232) 21.41.11;
(0232) 21.74.40 (difuzare); E-mail: office@polirom.ro
B-dul I.C. Brătianu nr. 6, et. 7, ap. 33; O.P. 37 • P.O. Box 1-728,
030174, Bucureşti; Tel.: (021) 313.89.78, E-mail: polirom@dnt.ro

Tiparul executat la S.C. LUMINA TIPO s.r.l.
str. Luigi Galvani nr. 20 bis, sect. 2, Bucureşti
Tel./Fax: (021) 211.32.60, (021) 212.29.27, E-mail: office@luminatipo.com

profesional
fundamente

Proiectarea bazelor de date

Normalizare
și postnormalizare

Implementări
SQL și Oracle

De la megaaplicațiile bancare, industriale sau militare până la banala agendă de telefon, bazele de date au devenit un ingredient indispensabil, chiar dacă mai puțin vizibil, al vieții de zi cu zi. Caracterul „discret", dar riguros conceptualizat și formalizat, la care se adaugă totuși o considerabilă doză de improvizație și intuiție au făcut din bazele de date un domeniu deosebit de atractiv pentru profesioniștii IT, proiectanții de aplicații și chiar simplii utilizatori. Adresată studenților, administratorilor de baze de date și programatorilor, cartea tratează, la nivel tehnic, conceptual și metodologic, beneficiind permanent și de exemple practice, problema delicată a proiectării unei baze de date. Practic, lucrarea prezintă cum s-ar putea „gândi" o schemă de baze de date pentru o aplicație anume, ce elemente ar trebui luate în considerare, chestiunile mai riguroase și cele mai flexibile, întregul demers fiind bazat pe normalizare și postnormalizare, precum și pe mecanismul de declarare a restricțiilor de integritate, coerență și consistență a datelor.

locul bazelor de date în aplicațiile software ▪ ce presupune proiectarea unei baze de date? ▪ a normaliza sau a nu normaliza? ▪ dependențe între atributele din bază ▪ câtă normalizare este necesară pentru a gândi o schemă acceptabilă a unei baze de date? ▪ punctele tari și cele nevralgice ale normalizării ▪ chestiuni teoretice legate de normalizare greu/imposibil de suportat ▪ chei surogat, dependențe de incluziune, ierarhii, generalizare și specializare ▪ cazuri practice de elaborare a schemei unei baze de date ▪ restricții complexe declarabile într-o bază de date ▪ modificarea schemei pentru creșterea vitezei de lucru ▪ denormalizare ▪ validitatea temporală a unei structuri relaționale ▪ protejarea datelor de modificări accidentale ulterioare ▪ comenzi SQL și blocuri Oracle PL/SQL pentru implementarea mecanismului de integritate

www.poliron

ISBN 973-681-898-5

POLIROM